

A Duality Based 2-Approximation Algorithm for Maximum Agreement Forest*

Frans Schalekamp¹, Anke van Zuylen², and Suzanne van der Ster³

¹School of Operations Research & Information Engineering, Cornell University, Ithaca, NY, fms9@cornell.edu

²Department of Mathematics, College of William & Mary, Williamsburg, VA, anke@wm.edu

³Institut für Informatik, Technische Universität München, Germany, ster@in.tum.de

April 29, 2016

Abstract

We give a 2-approximation algorithm for the Maximum Agreement Forest problem on two rooted binary trees. This NP-hard problem has been studied extensively in the past two decades, since it can be used to compute the Subtree Prune-and-Regraft (SPR) distance between two phylogenetic trees. Our result improves on the very recent 2.5-approximation algorithm due to Shi, Feng, You and Wang (2015). Our algorithm is the first approximation algorithm for this problem that uses LP duality in its analysis.

1 Introduction

Evolutionary relationships are often modeled by a rooted tree, where the leaves are a set of species, and internal nodes are (putative) common ancestors of the leaves below the internal node. Such phylogenetic trees date back to Darwin [5], who used them in his notebook to elucidate his thoughts on evolution.

The topology of phylogenetic trees can be based on different sources of data, e.g., morphological data, behavioral data, genetic data, etc., which can lead to different phylogenetic trees on the same set of species. Different measures have been proposed to measure the similarity of (or distance between) different phylogenetic trees on the same set of species (or individuals). Using the size of a maximum subtree common to both input trees as a similarity measure was proposed by Gordon [8]. The problem of finding such a subtree is now known as the Maximum Agreement Subtree Problem, and has been studied extensively. Steel and Warnow [13] are the first to give a polynomial-time algorithm for this problem. Their approach is refined to an $O(n^{1.5} \log n)$ -time algorithm by Farach and Thorup [6], who subsequently show their algorithm is optimal, unless unweighted bipartite matching can be solved in near linear time [7].

There exist non-tree-like evolutionary processes that preclude the existence of a phylogenetic tree, so-called reticulation events (such as hybridization, recombination and horizontal gene transfer). In this context, a particularly meaningful measure of comparing phylogenetic trees is the SPR-distance measure (where SPR is short for Subtree Prune-and-Regraft): this measure provides a lower bound on a certain type of these non-tree evolutionary events. The problem of finding the exact value of this measure for a set of species motivated the formulation of the Maximum Agreement Forest Problem (MAF) by Hein, Jian, Wang and Zhang [9]. Since the introduction by Hein et al., MAF has been extensively studied, including several variants, such as

*This work was initiated when the authors were visitors of Leen Stougie at the Tinbergen Institute. FS was supported in part by the Simon Prize for Excellence in the Teaching of Mathematics at William & Mary. AvZ was supported in part by a William & Mary Summer Research Award, NSF Prime Award: HRD-1107147, Women in Scientific Education (WISE) and by a grant from the Simons Foundation (#359525, Anke Van Zuylen).

the problem where the input consists of more than two trees, whether the input trees are rooted or unrooted, binary or non-binary. In our paper, we concentrate on MAF on two rooted binary trees.

For ease of defining the solutions to MAF on two rooted binary trees, we think of the input trees as being directed, where all edges are directed away from the root. Given two rooted binary trees on the same leaf set \mathcal{L} , the MAF problem asks to find a minimum set of edges to delete from the two trees, so that the directed trees in the resulting two forests can be paired up into pairs of “isomorphic” trees. Two trees are isomorphic if they contain the same nodes from \mathcal{L} and recursively removing nodes of out-degree zero that are not in \mathcal{L} and contracting two arcs incident to a node of in-degree and out-degree one, results in the same binary tree. An alternative (but equivalent) definition will be given in Section 2.

The problem of finding a MAF on two rooted binary trees has been extensively studied, although unfortunately some of the published results later turned out to have subtle errors. First of all, Allen and Steel [1] point out that the claim by Hein et al. that solving MAF on two rooted directed trees computes the SPR-distance between the trees is incorrect. Bordewich and Semple [4] show how to redefine MAF for rooted directed trees so that it is indeed the case that the optimal objective value of MAF is equal to the SPR-distance. In the paper in which they introduce MAF, Hein et al. [9] proved NP-hardness and they give an approximation algorithm for the problem, the approximation guarantee of which turned out to be slightly higher than what was claimed. Bordewich and Semple [4] show that, for their corrected definition of MAF, NP-hardness continues to hold. Other approximation algorithms followed [11, 2, 3]. The current best approximation ratio for MAF is 2.5, due to Shi, Feng, You and Wang [12], and Rodrigues [10] has shown that MAF is MAXSNP-hard. In addition, there is a body of work on other approaches, such as Fixed-Parameter Tractable (FPT) algorithms (e.g., [15, 14]) and Integer Programming [16, 17].

In this paper we give an improved approximation algorithm for MAF with an analysis based on linear programming duality. Our 2-approximation algorithm differs from previous works in two aspects. First of all, in terms of bounding the optimal value, we construct a feasible dual solution, rather than arguing more locally about the objective of the optimal solution. Secondly, our algorithm itself also takes a more global approach, whereas the algorithms in previous works mainly consider local substructures of at most four leaves. In particular, we identify a minimal subtree¹ of one of the two input trees for which the leaf set is incompatible, i.e., when deleting all other leaves from both trees, the remaining two trees are not isomorphic (such a minimal subtree can be found efficiently). We then use “local” operations which repeatedly look at two “sibling” leaves in the minimal incompatible subtree, and perform similar operations as those suggested by previous authors.

Preliminary tests were conducted using a proof-of-concept implementation of our algorithm in Java. Our preliminary results indicate that our algorithm finds a dual solution that in 44% of the 1000 runs is equal to the optimal dual solution, and in 37% of the runs is 1 less than the optimal solution. The observed average approximation ratio is about 1.92; following our algorithm with a simple greedy search algorithm decreases this to less than 1.28.

The remainder of our paper is organized as follows. In Section 2, we formally define the problem. In Section 3, we give a 3-approximation algorithm for MAF that introduces the dual linear program that is used throughout the remainder of the paper and gives a flavor of the arguments used to prove the approximation ratio of two. In Section 4, we give an overview of the 2-approximation algorithm and the key ideas in its analysis. In Section 7 we give more details on the randomly generated instances that we used to obtain our preliminary experimental results, and we conclude in Section 8 with some directions for future research.

2 Preliminaries

The input to the Maximum Agreement Forest problem (MAF) consists of two rooted binary trees T_1 and T_2 , where the leaves in each tree are labeled with the same label set \mathcal{L} . Each leaf has exactly one label, and each label in \mathcal{L} is assigned to exactly one leaf in T_1 , and one leaf in T_2 . For ease of exposition, we sometimes

¹By subtree of a rooted tree, we mean a tree containing the leaves that are descendants of some particular node in the rooted tree (including this node itself), and all edges between them.

think of the edges in the trees as being directed, so that there is a directed path from the root to each of the leaves.

We call the non-leaf nodes the internal nodes of the trees, and we let V denote the set of all nodes (internal nodes and leaves) in $T_1 \cup T_2$. Given a tree containing u and v , we let $\text{lca}(u, v)$ denote the lowest (closest to the leaves, furthest from the root) common ancestor of u and v . We let $\text{lca}_1(u, v)$ and $\text{lca}_2(u, v)$ denote $\text{lca}(u, v)$ in tree T_1 , respectively, T_2 . We extend this notation to $\text{lca}(U)$ which will denote the lowest common ancestor of a set of leaves U . For three leaves u, v, w and a rooted tree T , we use the notation $uv|w$ in T to denote that $\text{lca}(u, v)$ is a descendent of $\text{lca}(\{u, v, w\})$. A triplet $\{u, v, w\}$ of labeled leaves is *consistent* if $uv|w$ in $T_1 \Leftrightarrow uv|w$ in T_2 . The triplet is called *inconsistent* otherwise. We call a set of leaves $L \subseteq \mathcal{L}$ a *compatible set*, if it does not contain an inconsistent triplet.

For a compatible set $L \subseteq \mathcal{L}$, define $V[L] := \{v \in V : \text{there exists a pair of leaves } u, u' \text{ in } L \text{ so that } v \text{ is on the path between } u \text{ and } u' \text{ in } T_1 \text{ or } T_2\}$. Then, a partitioning L_1, L_2, \dots, L_p of \mathcal{L} corresponds to a feasible solution to MAF with objective value $p - 1$, if the sets L_1, L_2, \dots, L_p are compatible, and the sets $V[L_j]$ for $j = 1, \dots, p$ are node disjoint. Using this definition, we can write the following Integer Linear Program² for MAF: Let \mathcal{C} be the collection of all compatible sets of leaves, and introduce a binary variable x_L for every compatible set $L \in \mathcal{C}$, where the variable takes value 1 if the optimal solution to MAF has a tree with leaf set L . The constraints ensure that each leaf $v \in \mathcal{L}$ is in some tree in the optimal forest, and each internal node $v \in V \setminus \mathcal{L}$ is in at most one tree in the optimal forest. The objective encodes the fact that we need to delete $\sum_{L \in \mathcal{C}} x_L - 1$ edges from each of T_1 and T_2 to obtain forests with $\sum_{L \in \mathcal{C}} x_L$ trees.

$$\begin{array}{ll} \text{minimize} & \sum_{L \in \mathcal{C}} x_L - 1, \\ \text{s.t.} & \sum_{L: v \in L} x_L = 1 \quad \forall v \in \mathcal{L}, \\ & \sum_{L: v \in V[L]} x_L \leq 1 \quad \forall v \in V \setminus \mathcal{L}, \\ & x_L \in \{0, 1\} \quad \forall L \in \mathcal{C}. \end{array}$$

Remark The definition of MAF we use is *not* the definition that is now standard in the literature, but any (approximation) algorithm for our version can be used to get the same (approximation) result for the standard formulation: The standard formulation was introduced by Bordewich and Semple [4] in order to ensure that the objective value of MAF is equal to the rooted SPR distance. They note that for this to hold, we need the additional requirement that the two forests must also agree on the tree containing the original root; in other words, the original roots of T_1 and T_2 should be contained in a tree with the same (compatible) subset of leaves. An easy reduction shows that we can solve this problem using our definition of MAF: given two rooted binary trees for which we want to compute the SPR distance, we can simply add one new label ρ , and for each of the two input trees, we add a new root which has an edge to the original root and an edge to a new node with label ρ .³ A solution to “our” MAF problem on this modified input defines a solution to Bordewich and Semple’s problem on the original input with the same objective value and vice versa.

3 Duality Based 3-Approximation Algorithm

3.1 Algorithm

The algorithm we describe in this section is a variant of the algorithm of Rodrigues et al. [11] (see also Whidden and Zeh [15]). The algorithm maintains two forests, T'_1 and T'_2 on the same leaf set \mathcal{L}' , and iteratively deletes edges from these forests. At the start, T'_1 is set equal to T_1 , T'_2 to T_2 and \mathcal{L}' to \mathcal{L} . The leaves in \mathcal{L}' are called the *active* leaves. The algorithm will ensure that the leaves that are not active, will have been resolved in one of the two following ways: (1) they are part of a tree that contains only inactive leaves in both T'_1 and T'_2 ; these two trees then have the same leaf set, which is compatible, and they will be

²This ILP was obtained in discussions with Neil Olver and Leen Stougie.

³This is essentially the formulation that is common in the literature, except that in order to ensure that *only leaves have labels*, we give the label ρ to a new leaf that is an immediate descendent of the new root in both trees, rather than to the new root itself.

part of the final solution; or (2) an inactive leaf is *merged* with another leaf which is active, and in the final solution this inactive leaf will be in the same tree as the leaf it was merged with.

A tree is called *active* if it contains a leaf in \mathcal{L}' , and the tree is called *inactive* otherwise. An invariant of the algorithm is that there is a single active tree in T'_1 .

We define the parent of a set of active leaves W in a tree of a forest, denoted by $p(W)$, as the lowest node in the tree that is a common ancestor of W and at least one other active node. (That is, $p(W)$ is undefined if there are no other active leaves in the tree that contains the leaves in W .) Note that the parent of a node is defined with respect to the current state of the algorithm, and not with respect to the initial input tree. If $W = \{u\}$ is a singleton, we will also use the notation $p(u) = p(\{u\})$. For a given tree or forest T'_i , for $i \in \{1, 2\}$, we use the notation $p_i(W)$ to denote $p(W)$ in T'_i .

The operation in the algorithm that deletes edges from forest T'_i is *cut off a subset of leaves W in T'_i* . The edge that is deleted by this operation is the edge directly below $p_i(W)$ towards W (provided $p_i(W)$ is defined). Note that this means that the algorithm maintains the property that each internal node has a path to at least one leaf in \mathcal{L} . This ensures that the number of trees with leaves in \mathcal{L} in T'_i is equal to the number of edges cut in T'_i plus 1. It also ensures that the only leaves (nodes with outdegree 0) are the nodes in \mathcal{L} .

We will call two leaves u and v a *sibling pair* or *siblings* in a forest, if they belong to the same tree in the forest, and they are the only two leaves in the subtree rooted at the lowest common ancestor $\text{lca}(u, v)$. Similarly, u and v are an *active sibling pair* in a forest, if they belong to the same tree in the forest, and are the only two *active* leaves in the subtree rooted at the lowest common ancestor $\text{lca}(u, v)$ (an equivalent definition is that $p(u) = p(v)$ in the forest).

If leaves u and v are an active sibling pair in both T'_1 and T'_2 , we *merge* one of the leaves (say u) with the other (v). This means that from that point on v represents the subtree containing both u and v , instead of just the leaf v itself. This is accomplished by just making u inactive. Note that this merge operation can be performed recursively, where one or both of the leaves involved in the operation can already be leaves that represent subtrees. It is not hard to see that the subtree that is represented by an active leaf v is one of the two subtrees rooted at the child of $p(v)$, namely the subtree that contains v .

If leaves u and v are not active siblings in T'_2 (and they are active siblings in T'_1), we can choose to *cut off an active subtree* between leaves u and v . To describe this operation, let W_1, W_2, \dots, W_k be the active leaves of the active trees that would be created by deleting the path between u and v (both the nodes and the edges) in T'_2 . Note that $p_2(W_\ell)$ is on the path between u and v for all $\ell \in \{1, 2, \dots, k\}$, because u and v are not active siblings. Cutting off an active subtree between leaves u and v now means cutting off any such a set W_ℓ .

The algorithm is given in Algorithm 1. The boxed expressions refer to updates of the dual solution which will be discussed in Section 3.2.2. These expressions are only necessary for the analysis of the algorithm.

Theorem 1. *Algorithm 1 is a 3-approximation algorithm for the Maximum Agreement Forest problem.*

The proof of this theorem is given in the next subsection. It is clear the algorithm can be implemented to run in polynomial time. In Section 3.2.1, we show that the algorithm returns an agreement forest and we show that the number of edges deleted from T_2 by the algorithm can be upper bounded by three times the objective value of a feasible solution to the dual of a linear programming (LP) relaxation of MAF.

3.2 Analysis of the 3-Approximation Algorithm

3.2.1 Correctness

We need to show that the algorithm outputs an agreement forest. The trees of T'_1 and T'_2 each give a partitioning of \mathcal{L} , and clearly any internal node v belongs to $V[L]$ for at most one set in the partitioning. It remains to show that the two forests give the *same* partitioning of \mathcal{L} and that each set in the partitioning is compatible.

The algorithm ends with all trees in T'_1 and T'_2 being inactive, and the algorithm maintains that the set of leaves represented by an active leaf u (i.e., the leaves that were merged with u (recursively), and u itself) form the leaf set of a subtree in both T'_1 and T'_2 . To be precise, it is the subtree rooted at one of the children

```

1  $y_v \leftarrow 0$  for all  $v \in V$ .
2 while there exist at least 2 active leaves do
3   Find an active sibling pair  $u, v$  in the active tree in  $T'_1$ .
4   if  $u$  or  $v$  is the only active leaf in its tree in  $T'_2$  then
5     Cut off this node (say  $u$ ) in  $T'_1$  as well and make it inactive.  $y_u \leftarrow 1$ .
6   else
7     if  $u$  and  $v$  are active siblings in  $T'_2$  then
8       Merge  $u$  and  $v$  (i.e., make  $u$  inactive to “merge” it with  $v$ ).
9     else
10      if  $u$  and  $v$  are in the same tree in  $T'_2$ , and this tree contains an active leaf  $w$  such that  $uv|w$  in  $T_2$ 
11        then
12          Cut off an active subtree  $W$  between  $u$  and  $v$  in  $T'_2$ . Decrease  $y_{p_2(W)}$  by 1.
13        end if
14        Cut off  $u$  and cut off  $v$  in  $T'_1$  and  $T'_2$  and make them inactive.
15         $y_u \leftarrow 1, y_v \leftarrow 1$ , decrease  $y_{\text{lca}_1(u,v)}$  by 1.
16      end if
17    end if
18  end while
19 Make the remaining leaf (say  $v$ ) inactive.  $y_v \leftarrow 1$ .

```

Algorithm 1: A 3-Approximation for Maximum Agreement Forest. The boxed text refers to updates of the dual solution as discussed in Section 3.2.2.

of $p(u)$, namely the subtree that contains u . Furthermore note that this leaf set is compatible. This is easily verified by induction on the number of merges.

When u is the only active leaf in its tree in both forests, then the trees containing u in the two forests are thus guaranteed to have the same, compatible, set of leaves. Now, an inactive tree is created exactly when both T'_1 and T'_2 have an active tree in which some u is the only active leaf (lines 5, 13 and 18), and thus the two forests indeed induce the same partition of \mathcal{L} into compatible sets.

3.2.2 Approximation Ratio

In order to prove the claimed approximation ratio, we will construct a feasible dual solution to the dual of the relaxation of the ILP given in Section 2. The dual LP is given in Figure 1(a). The dual LP has an optimal solution in which $0 \leq y_v \leq 1$ for all $v \in \mathcal{L}$. The fact that $\{v\}$ is a compatible set implies that $y_v \leq 1$ must hold for every $v \in \mathcal{L}$. Furthermore, note that changing the equality constraints of the primal LP to \geq -inequalities does not change the optimal value, and hence we may assume $y_v \geq 0$ for $v \in \mathcal{L}$.

It will be convenient for our analysis to rewrite this dual by introducing additional variables for every (not necessarily compatible) set of labeled leaves. We will adopt the convention to use the letter A to denote a set of leaves that is not necessarily compatible, and the letter L to denote a set of leaves that is compatible (i.e., $L \in \mathcal{C}$). The dual LP can then be written as in Figure 1(b). Any solution to this new LP can be transformed into a solution to the original dual LP by, for each A such that $z_A > 0$, taking some leaf $v \in A$ and setting $y_v = y_v + z_A$ and $z_A = 0$. This is feasible because the left-hand side of the first family of inequalities will not increase for any compatible set L , and it will decrease for L such that $A \cap L \neq \emptyset$ and $v \notin L$. Conversely, a solution to the original dual LP is feasible for this new LP by setting $z_A = 0$, for every set of labeled leaves A .

We will refer to the left-hand side of the first family of constraints, i.e., $\sum_{v \in V[L]} y_v + \sum_{A: A \cap L \neq \emptyset} z_A$, as the *load* on set L .

Definition 1. The dual solution associated with a forest T'_2 , obtained from T_2 by edge deletions, active leaf

$$\begin{array}{ll}
\max & \sum_v y_v - 1, \\
\text{s.t.} & \sum_{v \in V[L]} y_v \leq 1 \quad \forall L \in \mathcal{C}, \\
& y_v \leq 0 \quad \forall v \in V \setminus \mathcal{L}.
\end{array}$$

(a) Dual LP

$$\begin{array}{ll}
\max & \sum_v y_v + \sum_{A \subseteq \mathcal{L}} z_A - 1, \\
\text{s.t.} & \sum_{v \in V[L]} y_v + \sum_{A: A \cap L \neq \emptyset} z_A \leq 1 \quad \forall L \in \mathcal{C}, \\
& y_v \leq 0 \quad \forall v \in V \setminus \mathcal{L}, \\
& y_v \geq 0 \quad \forall v \in \mathcal{L}, \\
& z_A \geq 0 \quad \forall A \subseteq \mathcal{L}.
\end{array}$$

(b) Reformulated dual LP

Figure 1: The dual of the LP relaxation for the ILP given in Section 2. The reformulated dual LP will be referred to as (D).

set \mathcal{L}' , and variables $y = \{y_v\}_{v \in V}$ is defined as (y, z) where $z_A = 1$ exactly when A is the active leaf set of a tree in T'_2 , and 0 otherwise. The objective value $\sum_v y_v + \sum_{A \subseteq \mathcal{L}} z_A - 1$ of the solution (y, z) associated with forest T'_2 , active leaf set \mathcal{L}' , and variables y will be denoted by $D(T'_2, \mathcal{L}', y)$.

We will sometimes use “the dual solution” to refer to the dual solution associated with T'_2, \mathcal{L}' and y when the forest, active leaf set and y -values are clear from the context.

Lemma 1. *After every execution of the while-loop of Algorithm 1, the dual solution associated with T'_2, \mathcal{L}' , and y is a feasible solution to (D) and the objective value $D(T'_2, \mathcal{L}', y)$ of this solution is at least $\frac{1}{3}|E(T_2) \setminus E(T'_2)|$.*

Proof. We prove the lemma by induction on the number of iterations. Initially, $z_{\mathcal{L}} = 1$, and all other variables are equal to 0. Clearly, this is a feasible solution with objective value 0.

Observe that the dual solution maintained by the algorithm satisfies that $y_u = 0$ while u is active. Therefore, if there is a single active leaf u in a tree in T'_2 , then making this leaf u inactive and setting $y_u = 1$ does not affect dual feasibility and the dual objective value, since making u inactive decreases $z_{\{u\}}$ from 1 to 0. Also note that merging two active leaves (and thus making one of the two leaves inactive), replaces the set of active leaves A in an active tree in T'_2 by a smaller set $A' \subset A$, with $A' \neq \emptyset$. Hence, the dual solution changes from having $z_A = 1$ to having $z_{A'} = 1$, which clearly does not affect dual feasibility or the dual objective value. Hence, we only need to verify that the dual solution remains feasible and its objective increases sufficiently for operations of the algorithm that cut edges from T'_2 , i.e., lines 11 and 13.

In line 11, one edge is cut in T'_2 , $y_{p_2(W)}$ decreases by 1. Let A be the set of active leaves in the tree containing W in T'_2 before cutting off W . z_A decreases by 1, $z_{A \setminus W}$ increases by 1, z_W increases by 1. The only sets L for which the left-hand side potentially increases are sets L so that $W \cap L \neq \emptyset$ and $(A \setminus W) \cap L \neq \emptyset$. However, $p_2(W) \in V[L]$ for such sets L , and since $y_{p_2(W)}$ is decreased by 1, the load is not increased for any compatible set L . The dual objective is unchanged, but will change in line 13 of the algorithm, as we will show next.

In line 13, let A_u be the set of active leaves in the tree in T'_2 containing u at the start of line 13 in the algorithm, and A_v be the set of active leaves in the tree in T'_2 containing v . Note that $A_u \setminus \{u, v\} \neq \emptyset$: if $v \notin A_u$, then this holds because otherwise we would execute line 5, and if $v \in A_u$, then this holds because u, v are not active siblings at the start of line 11, and if u, v became active siblings after executing line 11, then the condition for line 11 implies that there exists $w \in A_u$ such that $uw|w$ in T_2 .

The fact that $A_u \setminus \{u, v\} \neq \emptyset$ (and, by symmetry $A_v \setminus \{u, v\} \neq \emptyset$) implies that the total value of $\sum_A z_A + y_u + y_v$ increases by 2. Since we also decrease $y_{\text{lca}_1(u, v)}$ by 1 the total increase in the objective of the dual solution by line 13 is 1. Also, in lines 11 and 13, a total of at most three edges are cut in T'_2 .

It remains to show that executing line 13 does not make the dual solution (y, z) infeasible. Note that for each active set A' with $z_{A'} = 1$ before cutting off u and v , there is exactly one unique active subset $A \subseteq A'$ with $z_A = 1$ after cutting off u and v . Therefore the total value of $\sum_{A: A \cap L \neq \emptyset} z_A$ does not increase after cutting off u and v for any $L \subseteq \mathcal{L}$.

For any L that includes one of u, v , and at least one other active leaf, it must be the case that $\text{lca}_1(u, v) \in V[L]$, because all active leaves are in one tree in T'_1 , and u and v were active siblings in T'_1 at the start.

Hence the only compatible sets L for which the load on L potentially increases by 1 because of an increase in $\sum_{x \in L} y_x$ are sets L that include both u and v . We discern two cases.

Case 1: An active subtree W was cut off in line 11. In this case, the load on L was decreased by 1 in line 11, compensating for the increase in line 13: $V[L]$ contains all nodes on the path between u and v in T_2 , and hence also $p_2(W)$. It cannot contain a leaf $x \in W$, because $\{u, v, x\}$ form an inconsistent triplet (because $uv|x$ in T_1).

Case 2: No active subtree W was cut off in line 11. In this case, the value of $\sum_{A: A \cap L \neq \emptyset} z_A$ is decreased by at least 1: If u and v are in the same tree in T'_2 before cutting off u and v , then this tree contains no leaves x such that $uv|x$ in T_2 since otherwise an active subtree W would have been cut off. Hence, L does not contain any active leaf x in the active tree that remains after cutting off u and v in T'_2 , since any such leaf x does not have $uv|x$ in T_2 and therefore forms an inconsistent triplet with u and v . Since L does contain active leaves in the tree containing u and v in T'_2 before cutting off u and v (namely, u and v themselves), the value of $\sum_{A: A \cap L \neq \emptyset} z_A$ indeed decreases by 1.

If u and v are not in the same tree in T'_2 before cutting off u and v , then a similar argument holds. Since T'_2 is obtained from T_2 by deleting edges, at least one of the two active trees containing u and v contains no leaves x such that $uv|x$ in T_2 . Without loss of generality, suppose that this holds for the tree containing u . Then, L does not contain any active leaves in the active tree remaining after u is cut off, and hence $\sum_{A: A \cap L \neq \emptyset} z_A$ decreases by at least 1. \square

By weak duality, we have that the objective value of any feasible solution to (D) provides a lower bound on the objective value of any feasible solution to the LP relaxation of our ILP for MAF, and hence also on the optimal value of the ILP itself. Theorem 1 thus follows from Lemma 1 and the correctness shown in Section 3.2.1.

4 Overview of the 2-Approximation Algorithm

In this section, we begin by giving an outline of the key ideas of our 2-approximation algorithm. We then give an overview of the complete algorithm that we call the “Red-Blue Algorithm”.

One of the main ideas behind our 2-approximation is the consideration of the following two “essential” cases. The first “essential” case is the case where we have an active sibling pair u, v in T'_1 that are (i) active siblings in T'_2 , or (ii) in different trees in T'_2 , or (iii) the tree in T'_2 containing u, v does *not* contain an active leaf w such that $uv|w$ in T_2 . Then, it is easy to verify, using the arguments in the proof of Lemma 1, that Algorithm 1 “works”: it increases the dual objective value by at least half of the increase in $|E(T_2) \setminus E(T'_2)|$. We will say such a sibling pair u, v is a “success”.

The second “essential” case is the case where, in our current forest T'_1 , there is a subtree containing exactly three active leaves, say u, v, w , where $uv|w$ in T_1 , and $\{u, v, w\}$ is an inconsistent triplet; assume without loss of generality that $uw|v$ in T_2 , and that the first “essential” case does not apply; in particular, this implies that u, v are in the same tree in T'_2 . It turns out that such an inconsistent triplet can be “processed” in a way that allows us to increase the objective value of the dual solution in such a way that it “pays for” half the increase in the number of edges cut from T'_2 . There are a number of different cases to consider depending on whether all three leaves u, v, w are in the same tree in T'_2 (case I) or not (case II), and whether the tree in T'_2 containing w has an active leaf x such that $xw|u$ in T'_2 (case (a)) or not (case (b)).

Figure 2 gives an illustration of T'_1 and some possible configurations for T'_2 . Consider for example case (I)(b). Since $\{u, v, w\}$ is an inconsistent triplet, it is not hard to see that any solution to MAF either has v as a singleton component, or either u or w must be a singleton component. Indeed, we can increase the dual objective by 1, by updating the y -values as indicated by the circled values in Figure 2 (a) and (b). The bold diagonal lines denote two edges that are cut (deleted from T'_2). Similar arguments can be made for the other cases.

Unfortunately, neither of the essential cases may be present in the forests T'_1, T'_2 , and therefore the ideas given above may not be applicable. However, they do work if we generalize our notions. First, we generalize the notion of “active sibling pair in T'_1 ”.

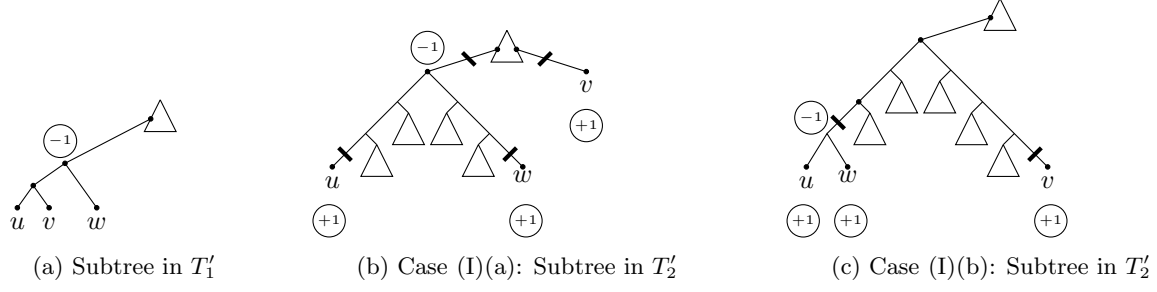


Figure 2: Dealing with an inconsistent triplet: Circled values denote the y -variables that are set by the algorithm, and bold diagonal lines denote edges that are cut (deleted from T'_2). Triangles denote subtrees with active leaves (that may be empty). Note that there is a distinction between edges that are incident to the root of a subtree represented by a triangle, and edges that are incident to some internal node of the subtree. The latter edges are connected to a dot on the triangle.

Definition 2. A set of active leaves U is an active sibling set in T'_1 if the leaves in U are the only active leaves in the subtree of T'_1 rooted at $\text{lca}_1(U)$. U is a compatible active sibling set in T'_1 if U is an active sibling set in T'_1 that contains no inconsistent triplets.

Note that we will only use the term compatible active sibling set for T'_1 , and never for T'_2 . We will therefore sometimes omit the reference to T'_1 , and simply talk about a “compatible active sibling set”.

We similarly generalize the notion of a subtree in T'_1 containing exactly three active leaves that form an inconsistent triplet.

Definition 3. A set of active leaves $R \cup B$ is a minimal incompatible active sibling set in T'_1 if $R \cup B$ is incompatible, R and B are compatible active sibling sets in T'_1 , and $p_1(R) = p_1(B)$.

The Red-Blue Algorithm now proceeds as follows: it begins by identifying a minimal incompatible active sibling set $R \cup B$ in T'_1 . Such a set can be found by checking if the active leaf sets of the left and right subtrees of the root are compatible sets. If yes, then either all active leaves are compatible, or we have found a minimal incompatible active set. If not, then the active leaf set of one of the subtrees is incompatible, and we recurse on this subtree until we find a node in T'_1 for which the active leaf sets of the left and right subtrees form a minimal incompatible set $R \cup B$. Note that we can assume $\text{lca}_2(R) = \text{lca}_2(R \cup B)$.

The algorithm will then “distill” R by repeatedly considering sibling pairs u, v in R , and executing operations similar to those in Algorithm 1, except that only one of u and v becomes inactive (and a bit more care has to be taken in certain cases). Procedure 1 gives the procedure RESOLVEPAIR the algorithm uses for handling a sibling pair u, v .

Arguments similar to those in Section 3 show that RESOLVEPAIR maintains dual feasibility, *provided that we initially reduce $y_{\text{lca}_1(R)}$ by 1*. It is also not hard to verify that RESOLVEPAIR increases the dual objective by at least half the increase in the primal objective, and the only thing that is therefore needed to show that the algorithm is a 2-approximation is that we can “make up for” the initial decrease of the dual objective caused by decreasing $y_{\text{lca}_1(R)}$. Let us define the operation of “distilling” R as starting by reducing $y_{\text{lca}_1(R)}$ by 1, and then repeatedly finding a pair of active leaves u, v in R which are siblings in T'_1 and executing RESOLVEPAIR(u, v) until only two active leaves \hat{u}, \hat{v} in R remain. Since all other leaves in R are inactive, \hat{u} and \hat{v} form an active sibling pair in T'_1 .

If pair \hat{u}, \hat{v} is a “success” or if line 4 or 15 was executed at least once during the distilling of R , then there exists an operation that makes at least one of \hat{u}, \hat{v} inactive and updates the dual solution, so that the total increase in the primal objective is at most twice the total increase in the dual objective caused by the processing of pairs in R . Procedure 2 gives the complete description of the procedure that, if successful, “resolves” set R (and will return “SUCCESS”):


```

1 if  $u$  and  $v$  are in different trees in  $T'_2$  then
2   Relabel  $u$  and  $v$  if necessary so that  $\text{lca}_2(u, v)$  is not in the tree containing  $u$  in  $T'_2$ .
3   if the tree containing  $u$  in  $T'_2$  has other active leaves not in  $U$  then
4     FinalCut: Cut off  $u$  in  $T'_1$  and  $T'_2$  and make it inactive.  $y_u \leftarrow 1$ .
5   else
6     Cut off  $u$  in  $T'_1$  and make it inactive.  $y_u \leftarrow 1$ .
7   end if
8 else
9   if  $u$  and  $v$  are active siblings in  $T'_2$  then
10    Merge  $u$  and  $v$  (i.e., make  $u$  inactive to “merge” it with  $v$ ).
11  else
12    Relabel  $u$  and  $v$  if necessary so that  $p_2(u) \neq \text{lca}_2(u, v)$ .
13    Cut off an active subtree  $W$  between  $u$  and  $v$  by cutting the edge below  $p_2(u)$  that is not on the path
        from  $u$  to  $v$ . Decrease  $y_{p_2(u)}$  by 1.
14    if  $u$  and  $v$  are now active siblings in  $T'_2$  then
15      Merge-After-Cut: Merge  $u$  and  $v$  (i.e., make  $u$  inactive to “merge” it with  $v$ ).  $y_u \leftarrow 1$ .
16    else
17      Cut off  $u$  in  $T'_1$  and  $T'_2$  and make  $u$  inactive.  $y_u \leftarrow 1$ .
18    end if
19  end if
20 end if

```

Procedure 1: RESOLVEPAIR(u, v)

Lemma 2. *If RESOLVESET(R) returns SUCCESS then at least one leaf in R became inactive, and the increase in the primal objective $|E(T_2) \setminus E(T'_2)|$ caused by the procedure is at most twice the increase in the dual objective.*

Lemma 5 in Section 5 contains a more precise formulation of this lemma.

If Lemma 2 applies, we have made progress (since we have made at least one leaf inactive), and we will have paid for the increase in the primal objective $|E(T_2) \setminus E(T'_2)|$ caused by the procedure by twice the increase in the dual objective.

Otherwise, the last active pair of leaves \hat{u}, \hat{v} in R remain active, and we will have a “deficit” in the sense that the increase in the dual objective is at most half the increase in the primal objective *plus* 1. In this case, we similarly distill B by repeatedly calling RESOLVEPAIR(u, v) for pairs u, v in B that are active siblings in T'_1 until only a single active leaf in B remains. However, we will show that in order to retain dual feasibility, we do not need to start the distilling of B by decreasing $y_{\text{lca}_1(B)}$ (which would give a total “deficit” of 2), but that we can “move” the initial decrease of $y_{\text{lca}_1(R)}$ to instead decrease $y_{\text{lca}_1(R \cup B)}$. Lemma 6 in Section 6 shows that this indeed preserves dual feasibility.

Once R and B have both been “distilled”, we are left with $\hat{u}, \hat{v}, \hat{w}$ that are an inconsistent triplet and form the active leaf set of a subtree in T'_1 . In Section 6, we then show how to deal with the triplet $\{\hat{u}, \hat{v}, \hat{w}\}$ (in ways similar to those in Figure 2) and we prove that in the entire processing of $R \cup B$, we have increased the dual objective by half of the number of edges we cut from T'_2 .

Algorithm 2 gives an overview of the “Red-Blue Algorithm”. It first calls a procedure PREPROCESS, which executes simple operations that do not affect the primal or dual objective: merging two leaves if they are active siblings in both forests, and cutting off and deactivating a leaf in T'_1 if it is the only active leaf in its tree in T'_2 . At the end of an iteration, the Red-Blue algorithm needs to consider different cases for the final triplet. The description of these subroutines can be found in Section 6.1 and Section 6.2.

Theorem 2. *The Red-Blue Algorithm is a 2-approximation algorithm for MAF.*

```

21 Decrease  $y_{\text{lca}_1(R)}$  by 1.
22 while there exist at least three active leaves in  $R$  do
23   | Find  $u, v$  in  $R$  that form an active sibling pair in  $T'_1$ .
24   |  $\text{RESOLVEPAIR}(u, v)$ .
25 end while
26 Let  $\hat{u}, \hat{v}$  be the remaining active leaves in  $R$ .
27 if  $\hat{u}$  and  $\hat{v}$  are active siblings in  $T'_2$  then
28   | Merge  $\hat{u}$  and  $\hat{v}$  (i.e., make  $\hat{u}$  inactive to “merge” it with  $\hat{v}$ ).  $y_{\hat{u}} \leftarrow 1$ .
29   | Return SUCCESS.
30 else if ( $\hat{u}$  and  $\hat{v}$  are in different trees in  $T'_2$ ) or (the tree containing  $\hat{u}$  and  $\hat{v}$  does not contain an active leaf  $w$  such that  $\hat{u}\hat{v}|w$  in  $T_2$ ) then
31   | Cut off  $\hat{u}$  in  $T'_2$  (if  $\hat{u}$ 's tree contains at least one other active leaf) and in  $T'_1$  and make  $\hat{u}$  inactive.  $y_{\hat{u}} \leftarrow 1$ .
32   | Cut off  $\hat{v}$  in  $T'_2$  (if  $\hat{v}$ 's tree contains at least one other active leaf) and in  $T'_1$  and make  $\hat{v}$  inactive.  $y_{\hat{v}} \leftarrow 1$ .
33   | Return SUCCESS.
34 else if (At least one FinalCut or Merge-After-Cut was executed in some call to  $\text{RESOLVEPAIR}$  in the course of the current  $\text{RESOLVESHET}$  procedure) then
35   |  $\text{RESOLVEPAIR}(\hat{u}, \hat{v})$ .
36   | Cut off the last active leaf  $\hat{v}$  in  $U$  in  $T'_2$  and in  $T'_1$  and make  $\hat{v}$  inactive.  $y_{\hat{v}} \leftarrow 1$ .
37   | Return SUCCESS.
38 else
39   | Return FAIL.
40 end if

```

Procedure 2: $\text{RESOLVESHET}(R)$

5 Distilling the Essence of a Compatible Active Sibling Set

In this section, we will prove (a more precise version of) Lemma 2, i.e., that if RESOLVESHET returns SUCCESS, then we have made progress towards a feasible primal solution, and we have increased the dual objective by at least half the increase in the primal objective. Because our arguments for $\text{RESOLVEPAIR}(u, v)$ will not only be used for a pair $u, v \in R$, but also (in Section 6) for a pair $u, v \in B$, we will let U denote an arbitrary compatible active sibling set U .

We begin by noting that our overall description of the algorithm is iterative, and that we thus assume that we have some global variables representing the forests T'_1 and T'_2 , the set of active leaves \mathcal{L}' and a setting of the dual variables y that are modified by the procedures.

Definition 4. We say a tuple $(T'_1, T'_2, \mathcal{L}', y)$ is valid for an instance (T_1, T_2, \mathcal{L}) of the Maximum Agreement Forest Problem, if

- T'_1 and T'_2 can be obtained from T_1 and T_2 respectively by edge deletions,
- all leaves in \mathcal{L}' are part of one tree in T'_1 ,
- $y_u = 0$ for all $u \in \mathcal{L}'$,
- the dual solution associated with T'_2 , \mathcal{L}' , and y is feasible for (D) ,
- the inactive trees in T'_1 and T'_2 can be paired up into pairs of trees with the same compatible leaf set,
- for each active leaf $u \in \mathcal{L}'$, the subtree containing u rooted at the child of $p_i(u)$ for $i = 1, 2$ contains the same leaf set in both forests, and this leaf set is compatible.

Note that $(T_1, T_2, \mathcal{L}, y_0)$ is valid for (T_1, T_2, \mathcal{L}) where $y_0(u) = 0$ for all leaves u .

In order to prove that a tuple remains valid after calling $\text{RESOLVEPAIR}(u, v)$, and in particular, that the associated dual solution remains feasible, we need one additional notion.

```

41 Set  $T'_1 \leftarrow T_1, T'_2 \leftarrow T_2, \mathcal{L}' \leftarrow \mathcal{L}$ .  $y_u \leftarrow 0$  for all  $u \in \mathcal{L}$ .
42 PREPROCESS.
43 while  $\mathcal{L}' \neq \emptyset$  do
44   Find a minimal incompatible active sibling set  $R \cup B$ , with  $\text{lca}_2(R) = \text{lca}_2(R \cup B)$ .
45   if RESOLVESET( $R$ ) returns FAIL then
46     Decrease  $y_{\text{lca}_1(R \cup B)}$  by 1, and increase  $y_{\text{lca}_1(R)}$  by 1.
47     while there exist at least two active leaves in  $B$  do
48       Find  $u, v$  in  $B$  that form an active sibling pair in  $T'_1$ .
49       RESOLVEPAIR( $u, v$ ).
50     end while
51     Let  $\hat{r}_1, \hat{r}_2 \in R$  and  $\hat{b} \in B$  be the remaining active leaves.
52     Consider three different cases depending on whether  $\hat{r}_1, \hat{r}_2$  and  $\hat{b}$  are in one, two or three different trees
      in  $T'_2$  (see Section 6.1 and 6.2 for details).
53   end if
54   PREPROCESS.
55 end while

```

Algorithm 2: Red-Blue Algorithm for Maximum Agreement Forest

Definition 5. For a compatible active sibling set $U \subseteq \mathcal{L}$, we call a tuple $(T'_1, T'_2, \mathcal{L}', y)$ U -safe, if for any compatible set of leaves L and any tree with active leaf set A in T'_2 the following holds: if $(L \cap A) \cap U \neq \emptyset$ and $(L \cap A) \setminus U \neq \emptyset$, then the load on L is at most 0 in the dual solution associated with T'_2, \mathcal{L}' and y .

The idea behind this definition is that the execution of RESOLVEPAIR on an active sibling pair u, v in T'_1 with $u, v \in U$ will only increase the load on sets L such that L contains both u and some leaf $w \notin U$ in the tree in T'_2 that contained u . Hence, U -safeness implies that the load remains at most 1 for compatible sets, and hence the dual solution remains feasible. We will furthermore show that U -safeness is preserved when u becomes inactive.

Observation 1. If U is an active sibling set in T'_1 , then we can make a given valid tuple U -safe, by decreasing $y_{\text{lca}_1(U)}$ or $y_{p_1(U)}$ by 1.

In RESOLVESET(U), we decrease $y_{\text{lca}_1(U)}$ as this will be helpful if the final active sibling pair \hat{u}, \hat{v} turns out to give a “success”, but the flexibility implied by Observation 1 will prove useful later if RESOLVESET fails; see Section 6.

The following lemma shows that U -safeness will ensure that RESOLVEPAIR returns a valid tuple, and that this tuple is again U -safe.

Lemma 3. Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple, let U be a compatible active sibling set in T'_1 , and let $u, v \in U$ be an active sibling pair in T'_1 . If $(T'_1, T'_2, \mathcal{L}', y)$ is U -safe, then the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ that results from the procedure RESOLVEPAIR(u, v) is a valid tuple that is U -safe.

Proof. The first three properties of a valid tuple are clear from the description of the procedure. The last two properties follow from the fact that only active sibling pairs in both T'_1 and T'_2 are merged, and trees become inactive when they contain a single active leaf, which is exactly the same as in Section 3.2.1. It remains to show that the modified dual solution is feasible and that it is U -safe.⁴

We consider the line numbers in an execution of RESOLVEPAIR that may affect the load on a set L :

4. (*FinalCut*) Let A be the active leaves in the tree in T'_2 containing u at the start of the procedure. Then z_A is decreased to 0, and y_u and $z_{A \setminus \{u\}}$ are increased to 1. This increases the load of a set L only if L contains u and an active leaf w in $A \setminus \{u\}$. Note that $w \notin U$ because $\text{lca}_2(u, w)$ must be in the tree

⁴In fact, we will prove a stronger property than U -safeness, namely, that the load on any compatible set L is at most 0 if $L \cap U$ contains active leaves. In the next section, we will see why we need the weaker definition of U -safeness.

containing u and w , and thus it would be a strict descendent of $\text{lca}_2(u, v)$, contradicting the fact that u, v are an active sibling pair in T'_1 and U is compatible. Therefore, by the fact that the input tuple is U -safe, the dual solution remains feasible.

It remains to prove that the new dual solution is U -safe. The load only increased for sets L that contain u and an active leaf w in $A \setminus \{u\}$. We will show that if L is compatible, then L cannot contain any other active leaf $u' \in U$, and hence, L does not need to have load at most 0 for the tuple to be U -safe. Suppose by contradiction that L contains $u, w \in A \setminus \{u\}$, and an active leaf $u' \in U$ with $u' \neq u$. Since U is a compatible active sibling set in T'_1 , $u'u|w$ in T_1 . On the other hand, $uw|v$ in T_2 , because $\text{lca}_2(u, v)$ is not on the path from u to w (by the condition in line 2), and thus also $uw|u'$ in T_2 because u, v is an active sibling pair in T'_1 and U is compatible. Hence $\{u, u', w\}$ is an inconsistent triplet and L is thus not compatible.

6. Since u is the only active leaf in its tree in T'_2 , $z_{\{u\}}$ decreases from 1 to 0. Therefore the load on any compatible set L does not increase when y_u is set to 1.
13. Let A be the set of active leaves in the tree containing W in T'_2 before cutting off W . z_A decreases by 1, $z_{A \setminus W}$ increases by 1, z_W increases by 1. The only sets L for which the load potentially increases by 1 are sets L so that $W \cap L \neq \emptyset$ and $(A \setminus W) \cap L \neq \emptyset$. However, $p_2(W) = p_2(u) \in V[L]$ for such sets L , and since $y_{p_2(u)}$ is decreased by 1, the load is not increased for any compatible set L , so the dual solution remains feasible and U -safe.
15. (*Merge-After-Cut*) Let A be the set of active leaves in the tree containing u and v . Since u becomes inactive, z_A decreases by 1, $z_{A \setminus \{u\}}$ increases by 1. y_u is set to 1. Therefore, if the load on a set L increases, then $u \in L$ and $L \cap (A \setminus \{u\}) \neq \emptyset$.

So let w be an active leaf in $L \cap (A \setminus \{u\})$. Note that $V[L]$ must contain the internal node that was $p_2(u)$ in line 13. Therefore, executing line 13 followed by line 15 only increases the load for L if it also contains an active leaf $w' \in W$, where W is the set that was cut off in line 13. Note that $uw'|w$ in T_2 , and that $u \in U$ and $w' \notin U$. Since L must be compatible, this implies that L cannot contain any active leaves in U after executing line 15, and, in particular, that $w \notin U$. Therefore, since u, w are both in A and the tuple was U -safe, the load on L becomes at most 1, and thus the dual solution remains feasible. Moreover, if L 's load increased then L contains no more active leaves in U , so the tuple remains U -safe.

17. Exactly the same arguments as for line 15 apply. □

Observation 2. Suppose the load on a set L increases during a call to $\text{RESOLVEPAIR}(u, v)$. Let u be the leaf among u and v that is deactivated in this call. Then it holds that $u \in L$, there exists $w \in L \setminus U$ so that $uw|v$ in T_2 , and $L \cap U$ does not contain any active leaves after the call to RESOLVEPAIR .

Proof. This follows directly from the proof of the previous lemma: the only lines where the load on any set L potentially increases are lines 4 and 13 followed by line 15 or 17. The observation follows directly from what is stated in the proof. □

We now consider the change in the objective value of the dual solution, i.e., $D(\tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y}) - D(T'_2, \mathcal{L}', y)$ and the objective value of the primal solution, i.e., $|E(T'_2) \setminus E(\tilde{T}'_2)|$. In Table 1 we use ΔD to denote the change in $D(T'_2, \mathcal{L}', y)$ caused by RESOLVEPAIR , and we denote by ΔP the change in $|E(T_2) \setminus E(T'_2)|$.

We see that each possibility increases the number of edges cut from T'_2 by at most twice the increase in the dual objective value. Furthermore, the two possibilities marked with a star (*FinalCut* and *Merge-After-Cut*) have $\Delta P \leq 2\Delta D - 1$.

Now, let $U = R$ be a compatible active sibling set, and consider $\text{RESOLVEMSET}(R)$ as given in Procedure 2. It starts by decreasing $\text{lca}_1(U)$ by 1 to make the tuple U -safe, and it then repeatedly calls RESOLVEPAIR , until only two active leaves in U remain. It continues to process these last two leaves only if it can guarantee that the total increase in the dual objective is at least $\frac{1}{2}$ times the increase in the number of edges cut from

line number	ΔP	ΔD
* 4 (<i>FinalCut</i>)	1	1
6	0	0
10	0	0
* 13 followed by 15 (<i>Merge-After-Cut</i>)	1	1
13 followed by 17	2	1

Table 1: ΔP denotes the change in $|E(T_2) \setminus E(T'_2)|$ and ΔD denotes the change in $D(T'_2, \mathcal{L}', y)$ for each possibility in Procedure 1: RESOLVEPAIR.

T'_2 ; in other words, if it can “make up” for the dual deficit that was created to make the initial tuple U -safe. In this case, the procedure outputs SUCCESS, and otherwise it outputs FAIL. In the latter case, the procedure terminates with two leaves in U still active.

We begin by showing that the tuple resulting from Procedure 2 is valid.

Lemma 4. *Procedure RESOLVESHET(U) executed on a compatible active sibling set U in T'_1 and a valid tuple $(T'_1, T'_2, \mathcal{L}', y)$ outputs a valid tuple $(T'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$.*

Proof. The first three properties of a valid tuple are again clear from the description of the procedure. The last two properties follow from the arguments in Section 3.2.1. It remains to show that the modified dual solution is feasible.

After executing line 21, the load on any set L that contains $\text{lca}_1(U)$ is decreased by 1, so the tuple is U -safe. By Lemma 3, the tuple is still valid and U -safe after completion of the while-loop.

At this moment in the procedure, \hat{u} and \hat{v} are the only active leaves remaining in U . It is easily verified that the remainder of the procedure increases the load only for sets containing \hat{u} or \hat{v} and at least one other active leaf. By U -safeness, we have that a compatible set L that contains \hat{u} or \hat{v} and least one active leaf $w \notin U$ will have load at most 0. For a compatible set L for which the active leaves are exactly \hat{u} and \hat{v} , the load will be at most 0 as well: Note that $\text{lca}_1(\hat{u}, \hat{v}) = \text{lca}_1(U)$ (because at every execution of the while-loop an active sibling pair from U is selected) and therefore, the load on a set L containing \hat{u} and \hat{v} is at most 0 after line 21. Because \hat{u} and \hat{v} are still active, it follows from Observation 2 that the load on any set containing \hat{u} or \hat{v} has not increased by calling RESOLVEPAIR.

Thus, to verify that the solution associated with T'_2, \mathcal{L}' and y will be a dual feasible solution at the end of the procedure, it remains to verify that lines 26–40 increase the load by at most 1 for any compatible set L .

If \hat{u}, \hat{v} are active siblings in T'_2 , then line 28 is the last line executed by the algorithm that changes \mathcal{L}' and y and clearly, the load on any set increases by at most 1. If lines 35–36 are executed, then by Lemma 3, the dual remains feasible and the load is at most 0 on sets L containing \hat{v} after executing line 35, and hence the dual will remain feasible after executing line 36 as well. Finally, suppose lines 31 and 32 are executed, and assume, by means of contradiction, that there is a compatible set L for which the load increases by more than 1. Let $A_{\hat{u}}$ be the active leaves in the tree containing \hat{u} before line 31 and let $A_{\hat{v}}$ be the active leaves in the tree containing \hat{v} (where it may be the case that $A_{\hat{u}} = A_{\hat{v}}$). L must contain \hat{v} and at least one active leaf $w \in A_{\hat{v}} \setminus \{\hat{v}\}$ (so that the load increases by 1 in line 31), and \hat{u} and at least one active leaf $w' \in A_{\hat{u}} \setminus \{\hat{u}, \hat{v}\}$ (so that the load increases by 1 in line 32). Now, if $A_{\hat{u}} = A_{\hat{v}}$, then the condition for line 31 implies that it cannot be the case that $\hat{u}\hat{v}|w'$ in T_2 , but this means $\{\hat{u}, \hat{v}, w'\}$ is an inconsistent triplet, contradicting the fact that L is compatible. If $A_{\hat{u}} \neq A_{\hat{v}}$, then at most one of $A_{\hat{u}}, A_{\hat{v}}$ can contain leaves q such that $\hat{u}\hat{v}|q$ in T_2 , and hence, either $\{\hat{u}, \hat{v}, w\}$ or $\{\hat{u}, \hat{v}, w'\}$ is an inconsistent triplet, and again, the fact that L is compatible is contradicted. \square

In order to show that, if the procedure outputs SUCCESS, the total increase in the dual objective is at least $\frac{1}{2}$ times the number of edges cut from T'_2 , we need to eliminate some “trivial” cases first, see Procedure 3.

Note that the processing in Procedure 3 cuts no edges from T'_2 , and that the merge operation in line 58 only merges active sibling pairs in both T'_1 and T'_2 , which is exactly the same as in Section 3.2.1. Dual

```

56 while (there exist active leaves  $u, v$  that form an active sibling pair in both  $T'_1$  and  $T'_2$ ) or (there exists an
    active leaf  $u$  that is the only active leaf in its tree in  $T'_2$ ) do
57   if there exist active leaves  $u, v$  that form an active sibling pair in both  $T'_1$  and  $T'_2$  then
58     Merge  $u$  and  $v$  (i.e., make  $u$  inactive to “merge” it with  $v$ ).
59   else
60     Let  $u$  be the only active leaf in its tree in  $T'_2$ .
61     Cut off  $u$  in  $T'_1$  (unless  $u$  is the last active leaf), and make  $u$  inactive.  $y_u \leftarrow 1$ .
62   end if
63 end while

```

Procedure 3: PREPROCESS

feasibility and the dual objective value are not affected by line 61, since $z_{\{u\}}$ is decreased by 1 when y_u is increased by 1.

Lemma 5. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple, that has been preprocessed by procedure PREPROCESS, and let U be an active sibling set in T'_1 . If RESOLVESET(U) returns SUCCESS, then it holds that*

$$D(\tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y}) - D(T'_2, \mathcal{L}', y) \geq \frac{1}{2}(|E(T'_2) \setminus E(\tilde{T}'_2)|),$$

for the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ that is output by RESOLVESET(U).

Proof. From Table 1, we see that the execution of each line increases the number of edges cut from T'_2 (ΔP) by at most twice the increase in the dual objective value ($2\Delta D$). If the line has a star, then $\Delta P = 2\Delta D - 1$. We also have that the first line of RESOLVESET, line 21, has $\Delta P = 2\Delta D + 2$, and we thus need to show that over the remainder of the procedure we “make up for” this initial decrease in the dual objective value by either executing two lines that have $\Delta P = 2\Delta D - 1$ or by executing a line that has $\Delta P = 2\Delta D - 2$.

If the algorithm returns SUCCESS, then it has either executed (a) line 28, or (b) lines 31 and 32, or (c) lines 35 and 36. In case (a), we are done, since line 28 had $\Delta P = 0$ and $\Delta D = 1$, so $\Delta P = 2\Delta D - 2$. In case (c), we are also done: note that when executing line 36, the last active leaf \hat{v} is in a tree in T'_2 with some leaf w such that $\hat{u}\hat{v}|w$ in T_2 (since otherwise, we would execute lines 31 and 32). Hence, cutting off \hat{v} in T'_2 gives $\Delta P = 1$ and $\Delta D = 1$ (and making \hat{v} inactive and setting $y_{\hat{v}} \leftarrow 1$ does not effect the dual objective). Since line 36 is only executed if at least one starred line from Table 1 was executed, we thus executed at least two lines that have $\Delta P = 2\Delta D - 1$ in total.

The only remaining case is the case when the algorithm terminates by executing lines 31 and 32 on pair \hat{u}, \hat{v} . Note that if \hat{u} and \hat{v} are in the same tree in T'_2 when executing lines 31 and 32, then this tree contains at least one other active leaf (since \hat{u} and \hat{v} are not siblings), and hence the last two lines together have $\Delta P = 2$ and $\Delta D = 2$, so $\Delta P = 2\Delta D - 2$. The subtle issue if \hat{u} and \hat{v} are not in the same tree in T'_2 is that if \hat{u} (or \hat{v}) is the only active leaf in its tree in T'_2 , then line 31 (respectively line 32) has $\Delta P = \Delta D = 0$, and hence this does not help to “make up for” the initial decrease in the dual objective value. If \hat{u} (or \hat{v}) is not the only active leaf in its tree in T'_2 , then line 31 (respectively line 32) has $\Delta P = 2\Delta D - 1$.

To analyze the case when \hat{u} and \hat{v} are in different trees in T'_2 , let U_L and U_R be the active leaf sets of the subtrees of T'_1 rooted at the two children of $\text{lca}_1(U)$ at the start of the procedure. Note that while there are at least three active leaves in $U = U_L \cup U_R$, an active sibling pair in T'_1 consisting of two leaves in U will contain either two leaves in U_R or two leaves in U_L . Therefore, each call to RESOLVEPAIR has as its arguments two leaves that are either both in U_L or both in U_R , and the last two leaves satisfy $\hat{u} \in U_L, \hat{v} \in U_R$.

We have the following claim.

Claim 1. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple that has been preprocessed using Procedure 3, and let \tilde{U} be a compatible active sibling set in T'_1 . If repeated calls to RESOLVEPAIR(u, v) with $u, v \in \tilde{U}$, where u, v are active siblings in T'_1 at the moment of the call, result in having a single active leaf \tilde{u} in \tilde{U} , which is the only active leaf in its tree in T'_2 , then a FinalCut or Merge-After-Cut must have been performed in one of the calls to RESOLVEPAIR.*

Note that the claim can be applied using $\tilde{U} = U_L$ and $\tilde{U} = U_R$ if \hat{u} , respectively \hat{v} , is the only active leaf in its tree in T'_2 . Hence, if \hat{u} and \hat{v} are in different trees in T'_2 , then both U_L and U_R contribute at least one operation that has $\Delta P = 2\Delta D - 1$ as required, and thus the total increase in the number of edges cut from T'_2 is at most twice the total increase in the dual objective value.

We conclude by proving the claim.

Proof of Claim: We will call an active tree in T'_2 \tilde{U} -unicolored, if all its active leaves are in \tilde{U} , and \tilde{U} -bicolored if it contains active leaves in \tilde{U} and active leaves not in \tilde{U} . Note that, initially, there must have been at least one active tree in T'_2 that was \tilde{U} -bicolored: otherwise, we could have preprocessed the tuple further in Procedure 3. Let A be the active leaf set of this tree. Now, either $\tilde{u} \in A$, or all leaves in $A \cap \tilde{U}$ will be inactive at the moment when \tilde{u} is the only active leaf remaining in \tilde{U} . It then follows from the following observation that at least one *FinalCut* or *Merge-After-Cut* must have been performed if the remaining leaf \tilde{u} is in a \tilde{U} -unicolored tree.

Observation 3. *Let A be the active leaf set of some active tree in T'_2 that is \tilde{U} -bicolored. If after a call to $\text{RESOLVEPAIR}(u, v)$ with $u, v \in \tilde{U}$, where u, v are active siblings in T'_1 , all leaves in $A \cap \tilde{U}$ are either inactive, or in a \tilde{U} -unicolored tree, then the RESOLVEPAIR procedure must have performed a *FinalCut* or *Merge-After-Cut*.*

To verify the observation, we consider the other possible executions of RESOLVEPAIR . Line 6 only deactivates a leaf that is in a \tilde{U} -unicolored tree. Line 10 deactivates a leaf in \tilde{U} and does not affect whether the tree containing this leaf is \tilde{U} -bicolored or not. Line 13 cuts off leaves that are not in \tilde{U} from the tree containing the leaves u and v . The remaining tree containing u and v is not \tilde{U} -unicolored, unless the procedure performs a *Merge-After-Cut*. \square

6 The Red-Blue Algorithm

In the previous section we showed a procedure to resolve certain compatible active sibling sets. The Red-Blue Algorithm (see Algorithm 2 in Section 4) uses this procedure as a subroutine: it starts by finding a “minimal incompatible active sibling set” $R \cup B$ and calls $\text{RESOLVESET}(R)$. In this section, we give more details on the Red-Blue Algorithm and a complete analysis of its correctness and approximation ratio.

First of all, note that we can always find a minimal incompatible active sibling set $R \cup B$ (if not all leaves are inactive after preprocessing using the PREPROCESS procedure): Consider $\text{lca}_1(\mathcal{L}')$. If the active leaf sets of the left and right subtree of this node are compatible, then let R and B be these two sets. Note that $R \cup B$ must be incompatible, since otherwise the PREPROCESS procedure would be able to make all leaves in $R \cup B$ inactive. If, on the other hand, the active leaf set of one of the subtrees is incompatible, we can recurse on this subtree until we find a node in T_1 for which the active leaf sets of the left and right subtrees are compatible.

We assume without loss of generality that $\text{lca}_2(R) = \text{lca}_2(R \cup B)$. A property that we will use in our analysis and that explains the distinction between sets R (the “red leaves”) and B (the “blue leaves”) is the following:

Observation 4. *Let $R \cup B$ be a minimal incompatible active sibling set such that $\text{lca}_2(R) = \text{lca}_2(R \cup B)$. Suppose $\text{RESOLVESET}(R)$ returns FAIL, and let \hat{r}_1, \hat{r}_2 be the remaining active leaves in R . Then*

1. $\{\hat{r}_1, \hat{r}_2, v\}$ is an inconsistent triplet for any $v \in B$,
2. \hat{r}_1, \hat{r}_2 are in the same tree in T'_2 ,
3. there exists an active leaf $w \notin R \cup B$ that is in the same tree in T'_2 as \hat{r}_1, \hat{r}_2 , where w is not a descendent of $\text{lca}_2(R \cup B)$ (i.e., w satisfies $uv|w$ in T_2 for all $u, v \in R \cup B$),
4. there exists an active leaf $x \notin R$ that is in the same tree in T'_2 as \hat{r}_1, \hat{r}_2 , where x is a descendent of $\text{lca}_2(R \cup B)$ (i.e., x satisfies $\hat{r}_1 x | \hat{r}_2$ in T_2 or $\hat{r}_2 x | \hat{r}_1$ in T_2).

Since $\text{lca}_2(\hat{r}_1, \hat{r}_2) = \text{lca}_2(R) = \text{lca}_2(R \cup B)$, we have that for any $v \in B$ it is not the case that $\hat{r}_1 \hat{r}_2 | v$ in T_2 , so $\{\hat{r}_1, \hat{r}_2, v\}$ is inconsistent. The fact that \hat{r}_1, \hat{r}_2 are in one tree in T'_2 follows from the fact that condition in line 30 of `RESOLVESET` did not hold since otherwise `RESOLVESET` would have returned `SUCCESS`. The existence of an active leaf w with the stated properties follows from the same fact: Note that the path in T'_2 connecting \hat{r}_1 and \hat{r}_2 contains $\text{lca}_2(R) = \text{lca}_2(R \cup B)$, and that the leaf w in line 30 is not a descendent of this node. Hence $uw | w$ in T_2 for every $u, v \in R \cup B$. The existence of x with the stated properties follows from the fact that the condition in line 27 did not hold.

If `RESOLVESET`(R) returns `FAIL`, then the Red-Blue Algorithm increases $\text{lca}_1(R)$ by 1 and decreases $\text{lca}_1(R \cup B)$. Note that by Observation 1, we could have initially made the tuple both R -safe and B -safe by decreasing $y_{p_1(R)} = y_{\text{lca}_1(R \cup B)}$ instead of $y_{\text{lca}_1(R)}$, so by Lemma 3, the tuple we have after `RESOLVESET`(R) fails is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe. The algorithm then proceeds to repeatedly call `RESOLVEPAIR`(u, v) for active sibling pairs $u, v \in B$ until a single active leaf \hat{b} in B remains. It follows from Observation 4 that the final three active leaves \hat{r}_1, \hat{r}_2 and \hat{b} form an inconsistent triplet, which form an active subtree in T'_1 . In Lemma 6 we show that the tuple we have at this moment is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe and $\{\hat{b}\}$ -safe. The next three subsections then explain how to deal with this triplet, depending on whether the leaves are all in one, two or three trees in T'_2 .

Lemma 6. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple at the start of line 44, and suppose `RESOLVESET`(R) returns `FAIL`. Then, the tuple in line 51 is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe and $\{\hat{b}\}$ -safe.*

Proof. Note that the fact that `RESOLVESET`(R) fails means that the only operations that have been executed are repeated calls to `RESOLVEPAIR`(u, v), with $u, v \in R$ or $u, v \in B$.

We first show that the tuple is valid, $\{\hat{r}_1, \hat{r}_2\}$ -safe and $\{\hat{b}\}$ -safe after executing line 46. We then show this continues to hold when executing lines 47–50.

By the fact that R is a compatible active sibling set in T'_1 , it follows from Lemma 3 that the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ resulting at the end of `RESOLVESET`(R) is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe. In fact, by Observation 2, the only sets L for which the load has increased in the course of `RESOLVESET`(R) are sets containing a leaf in R and a leaf not in R , and thus “moving up the -1 ” in line 46 does not affect the dual feasibility and the $\{\hat{r}_1, \hat{r}_2\}$ -safeness.

We now show that it is also $\{\hat{b}\}$ -safe. Note that the only operations that have been executed are calls to `RESOLVEPAIR`(u, v) with $u, v \in R$, and that the call did not perform a *FinalCut*, as in that case `RESOLVESET`(R) would return `SUCCESS`. We show that these operations cannot increase the load on a compatible set L that contains \hat{b} and at least one other active leaf $x \notin B$ that is in the same tree as \hat{b} in T'_2 . This proves that the tuple is $\{\hat{b}\}$ -safe after line 46, since executing this line decreases the load on any set containing \hat{b} and a leaf $x \notin B$.

Since we know the procedure did not execute a *FinalCut*, and since the load on L does not increase if u, v are active siblings in T'_2 , we only need to consider the case where the procedure executes line 13 followed by line 15 or line 17. Suppose by contradiction that L is compatible, the load on L increases, and L contains $\hat{b} \in B$ and an active leaf $x \notin B$ that are in the same tree in T'_2 after this operation. By Observation 2, if the load for L increases, then L contains u and a leaf $w \notin R$. Note that u and w are in an inconsistent triplet with any active leaf in R , because $uw | v$ in T_2 and u and v are active siblings in T'_1 . Therefore, it must be the case that $x \notin R$.

We discern three cases based on the relative position of $\text{lca}_2(u, b)$ and $\text{lca}_2(u, w)$ on the path from u to the root of T_2 .

Case 1: $\text{lca}_2(u, b) = \text{lca}_2(u, w)$. Note that $\text{lca}_2(x, b)$ must be a descendent of $\text{lca}_2(u, w)$, because the edge below $\text{lca}_2(u, w)$ towards w was cut, and x and b are still in the same tree in T'_2 . Thus $bx | u$ in T_2 , which contradicts that L is compatible because $x \notin R \cup B$.

Case 2: $\text{lca}_2(u, w)$ is a descendent of $\text{lca}_2(u, b)$. Then $uw | b$ in T'_2 , again contradicting that L is compatible because $w \notin R$.

Case 3: $\text{lca}_2(u, b)$ is a descendent of $\text{lca}_2(u, w)$. Then b is not in the same tree as u in T'_2 at the start of the procedure because $\text{lca}_2(u, w) = p_2(u)$. Therefore, $bx | u$ in T_2 , because x is in the same tree as b in T'_2 , again

contradicting that L is compatible because $x \notin R \cup B$.

It remains to consider the effect of executing $\text{RESOLVEPAIR}(u, v)$ for $u, v \in B$. Since B is a compatible active sibling set in T'_1 , by Lemma 3 the tuple remains valid, and by Observation 2 it does not increase the load on any set containing \hat{b} . We now consider the effect on a set L containing $r \in \{\hat{r}_1, \hat{r}_2\}$ and at least one other active leaf $x \notin \{\hat{r}_1, \hat{r}_2\}$ that is in the same tree as r in T'_2 . If $x \in B$, then the load on L did not increase by Observation 2, so assume instead that $x \notin B$. Moreover, if the load on a set L increases by executing $\text{RESOLVEPAIR}(u, v)$ for $u, v \in B$, then L must contain u and some leaf $w \notin B$ that was in the same tree as u at the start of $\text{RESOLVEPAIR}(u, v)$, and which satisfies $uw|v$ in T_2 .

We show that $\{u, r, w, x\}$ contains an inconsistent triplet by considering three cases. If r and u were in different trees in T'_2 after the execution of $\text{RESOLVASET}(R)$, then for any w that is in the same tree as u in T'_2 , it holds that $w \notin \{\hat{r}_1, \hat{r}_2\}$ (by Observation 4 (2)), and that $uw|r$ in T_2 , since $\text{lca}_2(u, r)$ must be on the path from r to the root, and hence $\text{lca}_2(u, r)$ is in the tree containing r and $\text{lca}_2(B \cup R)$ in T'_2 . Hence, in this case, $\{u, w, r\}$ form an inconsistent triplet, since $ur|w$ in T_1 . If r and u were in the same tree in T'_2 after the execution of $\text{RESOLVASET}(R)$, and they are still in the same tree in T'_2 after $\text{RESOLVEPAIR}(u, v)$, then $w \neq r$ and $uw|r$ in T_2 (since w must be a leaf in the set W that is cut off by cutting the edge below $p_2(u)$), and thus again $\{u, w, r\}$ is an inconsistent triplet. Finally, if r and u were in the same tree in T'_2 after the execution of $\text{RESOLVASET}(R)$, but some subsequent call to $\text{RESOLVEPAIR}(u', v')$ separates them in T'_2 , then r and x must be in the set W that is cut off by $\text{RESOLVEPAIR}(u', v')$. But then we have that $rx|u''$ in T_2 for any leaf $u'' \in B$ that is active at that time. Hence $rx|u$ in T_2 , and thus $\{r, x, u\}$ is an inconsistent triplet, since $ru|x$ in T_1 .

We have thus shown that the load cannot increase on a compatible set L that contains $r \in \{\hat{r}_1, \hat{r}_2\}$ and at least one other active leaf $x \notin \{\hat{r}_1, \hat{r}_2\}$ that is in the same tree as r in T'_2 . Therefore, the tuple remains $\{\hat{r}_1, \hat{r}_2\}$ -safe throughout lines 47–50. \square

6.1 Inconsistent triplet in a single tree in T'_2

Suppose \hat{r}_1, \hat{r}_2 and \hat{b} are in the same tree in T'_2 in line 51. Note that we are then exactly in case (I) of Section 4, and the triplet can either be in the configuration of subcase (I)(a) or of subcase (I)(b); see Figure 2. We give a formal description of the procedure for dealing with this case in Algorithm 4a.

Note that the execution of lines 47–50 can never delete an edge from T'_2 that is above $\text{lca}_2(R \cup B)$, because $\text{RESOLVEPAIR}(u, v)$ only cuts edges below $\text{lca}_2(u, v)$, and hence in lines 47–50 only edges below $\text{lca}_2(B)$ are cut. Combined with Observation 4, this implies that the tree in T'_2 containing \hat{r}_1, \hat{r}_2 and \hat{b} contains at least one active leaf $w \notin R \cup B$ that is not a descendent of $\text{lca}_2(R \cup B)$.

64	Relabel \hat{r}_1, \hat{r}_2 if necessary so that $\hat{b}\hat{r}_1 \hat{r}_2$ in T_2 .	
65	Cut off the subtree rooted at $\text{lca}_2(\hat{r}_1, \hat{b})$ in T'_2 .	Decrease $y_{\text{lca}_2(\hat{r}_1, \hat{b})}$ by 1.
66	Cut off \hat{r}_2 in T'_2 and T'_1 and make \hat{r}_2 inactive.	$y_{\hat{r}_2} \leftarrow 1$.
67	if \hat{r}_1, \hat{b} are now active siblings in T'_2 then	
68	Merge \hat{b} and \hat{r}_1 (i.e., make \hat{b} inactive to “merge” it with \hat{r}_1).	$y_{\hat{b}} \leftarrow 1$.
69	else	
70	Cut off \hat{r}_1 and \hat{b} in T'_2 and T'_1 and make them inactive.	$y_{\hat{r}_1} \leftarrow 1, y_{\hat{b}} \leftarrow 1$.
71	end if	

Procedure 4a: \hat{r}_1, \hat{r}_2 and \hat{b} are in the same tree in T'_2

Lemma 7. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple, that has been preprocessed by procedure PREPROCESS , and let $R \cup B$ be a minimal incompatible active sibling set with $\text{lca}_2(R) = \text{lca}_2(R \cup B)$, for which $\text{RESOLVASET}(R)$ returns FAIL. If, after executing lines 44–50, the three remaining active leaves are in a single tree in T'_2 , then*

the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ after executing lines 44–50 followed by Procedure 4a is valid, and satisfies

$$D(\tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y}) - D(T'_2, \mathcal{L}', y) \geq \frac{1}{2}(|E(T'_2) \setminus E(\tilde{T}'_2)|).$$

Proof. The first three properties of a valid tuple are again clear from the description of the procedure. The last two properties follow from the arguments in Section 3.2.1. It remains to show that the modified dual solution is feasible, and that the increase in the dual objective value can “pay for” half of the increase in the primal objective value.

Letting, as in line 51, \hat{r}_1, \hat{r}_2 be the remaining active leaves in R and \hat{b} the remaining active leaf in B , we have by Lemma 6 that the tuple is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe and $\{\hat{b}\}$ -safe after lines 44–50.

We consider what happens to the dual solution when executing Procedure 4a. The numbers refer to the line numbers in the procedure.

65. Let A_1, A_2 be the active leaf sets of the two trees created, with $\hat{b}, \hat{r}_1 \in A_1, \hat{r}_2 \in A_2$. Then $z_{A_1 \cup A_2}$ decreases by 1 and z_{A_1} and z_{A_2} increase by 1. Note that $y_{\text{lca}_2(\hat{r}_1, \hat{b})}$ is decreased by 1. Since any set L with $L \cap A_1 \neq \emptyset$ and $L \cap A_2 \neq \emptyset$ has $\text{lca}_2(\hat{r}_1, \hat{b}) \in V[L]$, this therefore does not increase the load on any set L . The objective value of the dual solution is also not changed.
66. Let A_2 be the active leaves of the tree in T'_2 containing \hat{r}_2 before line 66 is executed. Note that $A_2 \setminus \{\hat{r}_2\}$ is not empty, because it contains a node that is not a descendent of $\text{lca}_2(R \cup B)$ (see Observation 4). Therefore line 66 increases $z_{A_2 \setminus \{\hat{r}_2\}}$ by 1; it also decreases z_{A_2} by 1 and increases $y_{\hat{r}_2}$ by 1. This increases the load on sets L containing \hat{r}_2 and at least one leaf $w \in A_2 \setminus \{\hat{r}_2\}$. Note that $w \neq \hat{r}_1$, since $\hat{r}_1 \notin A_2$. By the fact that the dual solution is $\{\hat{r}_1, \hat{r}_2\}$ -safe, we thus know that the load on any set L for which the load increases had load at most 0 prior to the increase. The dual objective value is increased by 1.
68. If \hat{b} and \hat{r}_1 are active siblings in T'_2 , they are the only active leaves in their tree in T'_2 . Hence, line 68 decreases $z_{\{\hat{r}_1, \hat{b}\}}$ by 1, and it increases $y_{\hat{b}}$ and $z_{\{\hat{r}_1\}}$ by 1. The load is increased only on sets L containing both \hat{b} and \hat{r}_1 . Such sets L had load at most 0 at the start of the procedure by Lemma 6. Furthermore, L cannot have had its load increased in line 66, as this would mean L contains inconsistent triplet $\{\hat{b}, \hat{r}_1, \hat{r}_2\}$. Hence, the dual solution remains feasible. The dual objective value is increased by 1.
70. The value of z_{A_1} is decreased by 1, and $y_{\hat{b}}, y_{\hat{r}_1}$ and $z_{A_1 \setminus \{\hat{b}, \hat{r}_1\}}$ are increased by 1. This increases the load on sets L containing leaves in at least two of the sets $\{\hat{b}\}, \{\hat{r}_1\}, A_1 \setminus \{\hat{b}, \hat{r}_1\}$. Furthermore, note that a compatible set L can contain leaves in at most two of these sets, and thus the load on a compatible set L increases by at most 1.

If the load on a compatible set L increases, then L cannot contain \hat{r}_2 : the load on L increasing implies that L contains \hat{b} or \hat{r}_1 , and at least one other leaf in A_1 . These two nodes, say u, v , in A_1 and \hat{r}_2 are an inconsistent triplet: $uv|\hat{r}_2$ in T_2 , but u, v are not both in R and they are not both in B , so it cannot be the case that $uv|\hat{r}_2$ in T_1 . This shows that L is incompatible.

Therefore the load on L was at most 0 at the start of the procedure (by Lemma 6) and the load has not increased by line 66. Hence, the dual solution remains feasible. The dual objective value is increased by 2.

We now consider the total change in the primal and dual objective value. Let ΔP_1 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to lines 44–50, and let ΔP_2 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to Procedure 4a. Similarly, let ΔD_1 be the total change in the dual objective value by lines 44–50, and ΔD_2 the change in the dual objective due to Procedure 4a.

We have $\Delta D_1 \geq \frac{1}{2}\Delta P_1 - 1$ by taking into account the initial decrease in the dual objective value and Table 1. Note that $\Delta P_2 = 2$ if line 68 is executed, and $\Delta P_2 = 4$ if line 70 is executed. The arguments about the dual solution given above also show that $\Delta D_2 = 2$ in the first case, and $\Delta D_2 = 3$ in the second case. Hence, we have that $\Delta D_2 \geq \frac{1}{2}\Delta P_2 + 1$, and thus $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}(\Delta P_1 + \Delta P_2)$. \square

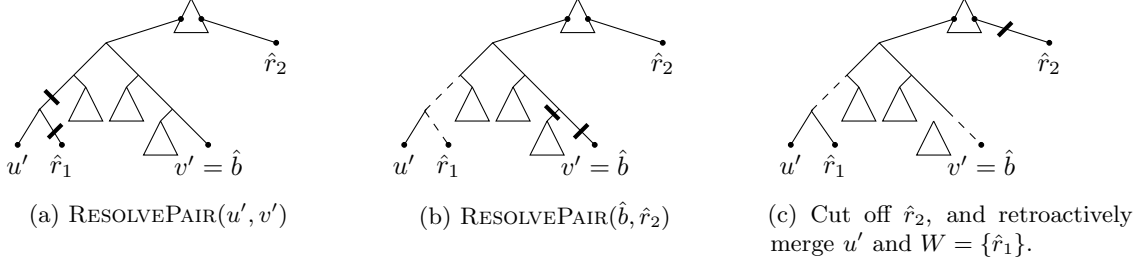


Figure 3: Illustration of a case where a retroactive merge is needed. The set R contains leaves \hat{r}_1, \hat{r}_2 only, and the set B contains two leaves, u' and v' (where v' will be the last remaining active leaf in B , so $v' = \hat{b}$). Figure (a) shows the execution of $\text{RESOLVEPAIR}(u', v')$. After this, we execute Procedure 4c, with $\hat{u} = \hat{r}_1$, $\{\hat{v}_1, \hat{v}_2\} = \{\hat{b}, \hat{r}_2\}$.

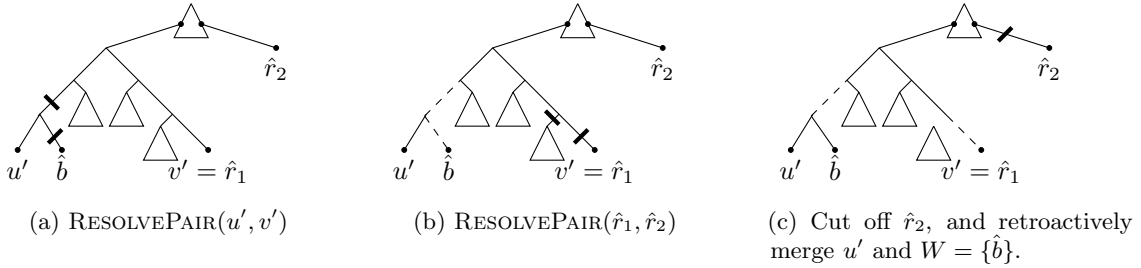


Figure 4: Illustration of a second case where a retroactive merge is needed. The set R contains three leaves u', \hat{r}_1, \hat{r}_2 , and the set B contains a single leaf, \hat{b} . Figure (a) shows the execution of $\text{RESOLVEPAIR}(u', v')$, with $v' = \hat{r}_1$. After this, we execute Procedure 4c, with $\hat{u} = \hat{b}$, $\{\hat{v}_1, \hat{v}_2\} = \{\hat{r}_1, \hat{r}_2\}$.

6.2 Inconsistent triplet in multiple trees in T'_2

We give the procedures for dealing with the remaining cases in Procedure 4b and Procedure 4c. These are more complicated than case (II)(a) that was shown in Figure 2 in Section 4. The reason for this additional complexity is that the calls to $\text{RESOLVEPAIR}(u, v)$ with $u, v \in R \cup B$, may lead to \hat{r}_1, \hat{r}_2 or \hat{b} being the only active leaf in their respective trees in T'_2 . If this happens, we may need to identify two inactive trees at the end of the procedure and “retroactively merge” them. In Figures 3 and 4, we give examples that illustrate the retroactive merge for Procedure 4c.

The analysis for the two procedures overlaps in part. In the current subsection, we will give these parts of the analysis, including an explanation of what we mean by retroactively merging W and u' in the last line of the two procedures. The proofs in this section are quite technical, and to maintain the flow of the argument, they have been deferred to Section A.

In Section 6.2.1 and Section 6.2.2, we show that a pass through the while-loop that goes through Procedure 4b and Procedure 4c, respectively, outputs a valid tuple, and increases the dual objective value by at least half of the number of edges cut from T'_2 .

Our first claim implies that in line 72 and line 89 the dual objective value will increase by 1.

Claim 2. *Just before line 72 and line 89, the leaf \hat{b} and \hat{v}_2 , respectively, is not the only active leaf in its tree in T'_2 .*

The difficulty in Procedures 4b and 4c lies in the case where the condition in line 75 or line 90, respectively, evaluates to true. In this case, the dual objective in the current pass through the while-loop has not increased enough to “pay for” (half of) the edges that were deleted from T'_2 .

72 Cut off \hat{b} in T'_2 and T'_1 and make \hat{b} inactive. $y_{\hat{b}} \leftarrow 1.$ See Claim 2.
 73 Cut off \hat{r}_1 in T'_2 (if \hat{r}_1 's tree contains at least one other active leaf) and in T'_1 and make \hat{r}_1 inactive. $y_{\hat{r}_1} \leftarrow 1.$
 74 Cut off \hat{r}_2 in T'_2 (if \hat{r}_2 's tree contains at least one other active leaf) and in T'_1 and make \hat{r}_2 inactive. $y_{\hat{r}_2} \leftarrow 1.$
 75 **if** (\hat{r}_1 and \hat{r}_2 were each the only active leaf in its tree in T'_2 in lines 73–74) and (no FinalCut or Merge-After-Cut was executed in some call to RESOLVEPAIR in lines 47–50) **then**
 76 Find the inactive leaves $u', v' \in B$ for which the execution of RESOLVEPAIR(u', v') cut off set $W \ni \hat{r}_2$ in line 13. Let u' be the leaf that was deactivated by RESOLVEPAIR(u', v'). Retroactively merge the trees containing W and u' (see Claims 3 and 4, and the remarks following these).
 77 **end if**

Procedure 4b: \hat{r}_1, \hat{r}_2 and \hat{b} are in three distinct trees in T'_2

78 Relabel $\hat{r}_1, \hat{r}_2, \hat{b}$ as $\hat{u}, \hat{v}_1, \hat{v}_2$, so that \hat{u} is the only active leaf in $R \cup B$ in its tree in T'_2 , and \hat{v}_1, \hat{v}_2 are in the same tree in T'_2 .
 79 **if** \hat{u} is the only active leaf in its tree in T'_2 **then**
 80 Cut off \hat{u} in T'_1 and make u inactive. $y_{\hat{u}} \leftarrow 1.$
 81 **else**
 82 Cut off \hat{u} in T'_2 and T'_1 and make u inactive. $y_{\hat{u}} \leftarrow 1.$
 83 **end if**
 84 **if** \hat{v}_1, \hat{v}_2 are active siblings in T'_2 **then**
 85 Relabel \hat{v}_1 and \hat{v}_2 if necessary so $\hat{v}_1 = \hat{b}$. See Claim 5.
 86 Merge \hat{v}_1 and \hat{v}_2 (i.e., make \hat{v}_1 inactive to “merge” it with \hat{v}_2). $y_{\hat{b}} \leftarrow 1.$
 87 **else**
 88 RESOLVEPAIR(\hat{v}_1, \hat{v}_2).
 89 Cut off the last active leaf in $R \cup B$, say \hat{v}_2 , in T'_2 and T'_1 and make \hat{v}_2 inactive. $y_{\hat{v}_2} \leftarrow 1.$ See Claim 2.
 90 **if** (line 80 was executed) and (no FinalCut or Merge-After-Cut was executed in line 88 or lines 47–50) **then**
 91 Find the inactive leaves $u', v' \in R \cup B$ for which the execution of RESOLVEPAIR(u', v') cut off set $W \ni \hat{u}$ in line 13. Let u' be the leaf that was deactivated by RESOLVEPAIR(u', v'). Retroactively merge W and u' (see Claims 3 and 4, and the remarks following these).
 92 **end if**
 93 **end if**

Procedure 4c: \hat{r}_1, \hat{r}_2 and \hat{b} are in two distinct trees in T'_2

Luckily, it turns out that will be able to “retroactively merge” two inactive trees in the two forests in this case. The next claim allows us to identify two inactive trees, one for which the leaves that were active at the start of the current pass through the while-loop are in R and one for which the leaves that were active at the start of the while-loop are in B . Furthermore, we will be able to identify a single call to `RESOLVEPAIR` in the algorithm in which these two trees were both cut off from T'_2 .

In the following claim, we say that two leaves have the same color, if they are in the same set, either R or B .

Claim 3. *The leaves u', v' in line 76 and line 91 exist. Furthermore, the set W in line 76 and line 91 contains only active leaves that have the same color as \hat{r}_2 and \hat{u} , respectively.*

Note that either $u' \in R$ and $W \subseteq B$, or $u' \in B$ and $W \subseteq R$. In the latter case, we must have that W is a singleton, since the operation of `RESOLVEPAIR`(u', v') must have occurred after completing `RESOLVEMERGE`(R). In the former case, the active leaves in W will be merged by calls to `RESOLVEPAIR` until only \hat{u} remains. Retroactively merging W and u' means that we want to restore the paths connecting them in both forests. In order to retroactively merge them, we thus need to show that nodes on the paths between these two trees in T'_1 and T'_2 are not “used” to connect two leaves in another active or inactive tree. This is what is shown in the proof of the following claim. For a set of leaves L , we let $V_1[L]$ be the nodes in $V[L]$ that are in T_1 , and we let $V_2[L]$ be the nodes in $V[L]$ that are in T_2 .

Claim 4. *Let u' be the leaf and W the set identified in line 76 or line 91, respectively, and let L_1, \dots, L_k be the leaf sets of the trees in T'_1 at this moment, excluding the trees containing u' and W , and let $M_1, \dots, M_{k'}$ be the leaf sets of the trees in T'_2 at this moment, excluding the trees containing u' and W . Then, $V_1[L_j] \cap V_1[\{u'\} \cup W] = \emptyset$ for all $j = 1, \dots, k$ and $V_2[M_{j'}] \cap V_2[\{u'\} \cup W] = \emptyset$ for all $j' = 1, \dots, k'$.*

By the claim, we can merge the trees containing u' and W : we reinsert the missing edges connecting the leaves in $V[\{u'\} \cup W]$ in the two forests, each time adding an edge with one endpoint in the tree containing W and the other endpoint in $V[\{u'\} \cup W]$ but not in the tree containing W . Whenever the addition (undeletion) of an edge merges the tree containing W with L_j or $M_{j'}$ for some j or j' , there must be a node in $V[\{u'\} \cup W]$ that is incident to an edge, for which the other endpoint is not in $V[\{u'\} \cup W]$. We delete the latter edge, and continue. By the claim, this edge exists, and deleting this edge does not disconnect L_j or $M_{j'}$.

Since the number of trees in both forests decreases by 1 (since the edge that connects the tree containing W and the tree containing u' does not lead to a deleted edge), we have thus shown that the retroactive merge decreases $|E(T_2) \setminus E(T'_2)|$ by 1.

6.2.1 Analysis of Procedure 4b

Lemma 8. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple, that has been preprocessed by procedure `PREPROCESS`, and let $R \cup B$ be a minimal incompatible active sibling set with $\text{lca}_2(R) = \text{lca}_2(R \cup B)$, for which `RESOLVEMERGE`(R) returns `FAIL`. If, after executing lines 44–50, the three remaining active leaves are in three distinct trees in T'_2 , then the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ after executing lines 44–50 followed by Procedure 4b is valid, and satisfies*

$$D(\tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y}) - D(T'_2, \mathcal{L}', y) \geq \frac{1}{2}(|E(T'_2) \setminus E(\tilde{T}'_2)|).$$

Proof. It follows from the remarks about the retroactive merge at the end of the previous subsection that the Procedure 4b maintains all properties of a valid tuple, except possibly for the feasibility of the modified dual solution. In addition, we need to show that the increase in the dual objective value can “pay for” half of the increase in the primal objective value.

As before, we know by Lemma 6 that the tuple is valid and $\{\hat{r}_1, \hat{r}_2\}$ -safe and $\{\hat{b}\}$ -safe at the start of Procedure 4b. We go through the lines of Procedure 4b and consider the effect on the dual solution.

72. By Claim 2, the active leaf set A of the tree in T'_2 containing \hat{b} before executing line 72 contains at least one leaf in addition to \hat{b} . Executing this line, decreases z_A by 1 and increases $z_{A \setminus \{\hat{b}\}}$ and $y_{\hat{b}}$ by 1.

This increases the load on compatible sets L containing \hat{b} and some leaf in $A \setminus \{\hat{b}\}$, and by $\{\hat{b}\}$ -safeness,

the load on these sets was at most 0. Hence, the dual solution remains feasible. The objective value of the dual solution increases by 1.

73–74. Let A_1 and A_2 be the active leaves of the tree in T'_2 containing \hat{r}_1 and \hat{r}_2 , respectively, before executing lines 73–74. The effect of these lines on the dual solution depends on whether A_1, A_2 contain other leaves or not. If they do not, then the dual solution is effectively unchanged, since if $A_i = \{\hat{r}_i\}$, we simply decrease z_{A_i} by 1 and we increase $y_{\hat{r}_i}$ by 1. If A_i contains some leaf in addition to \hat{r}_i , we also increase $z_{A_i \setminus \{\hat{r}_i\}}$ by 1.

Note that the load on a compatible set L increases by these lines, if L contains \hat{r}_i and a leaf in $A_i \setminus \{\hat{r}_i\}$ for i equal to 1 and/or 2. By the fact the dual solution was $\{\hat{r}_1, \hat{r}_2\}$ -safe at the start of the procedure, we know that the load on L was at most 0 at the start of the procedure.

Furthermore, we claim that a compatible set L can get an increase in its load from only one of the three “splits”, i.e., either because it contains \hat{b} and a leaf in $A \setminus \{\hat{b}\}$, or because it contains \hat{r}_1 and a leaf in $A_1 \setminus \{\hat{r}_1\}$, or because it contains \hat{r}_2 and a leaf in $A_2 \setminus \{\hat{r}_2\}$. This is because at most one of the trees in T'_2 at the start of the procedure contains $\text{lca}_2(R \cup B)$, and hence at most one of $A \setminus \{\hat{b}\}$, $A_1 \setminus \{\hat{r}_1\}$ and $A_2 \setminus \{\hat{r}_2\}$ contains leaves x such that $uv|x$ for $u, v \in R \cup B$. If L contains a leaf x such that $uv|x$ does not hold for $u, v \in R \cup B$, then it cannot contain two leaves in $R \cup B$.

Thus, the load increases by at most 1 on any compatible set L , and a set for which the load increases had load 0 at the start of the procedure. The remaining lines do not affect the dual solution, and hence, we have shown that the dual solution remains feasible.

We now consider the total change in the primal and dual objective value. Let ΔP_1 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to lines 44–50, and let ΔP_2 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to lines 72–74 of Procedure 4b. Similarly, let ΔD_1 be the total change in the dual objective value by lines 44–50, and ΔD_2 the change in the dual objective due to Procedure 4b.

As in the proof of Lemma 7, we have $\Delta D_1 \geq \frac{1}{2}\Delta P_1 - 1$. Furthermore, we know that if at least one *FinalCut* or *Merge-After-Cut* is executed, that $\Delta D_1 \geq \frac{1}{2}\Delta P_1 - 1 + \frac{1}{2}$.

Each edge deleted from T'_2 by lines 72–74 contribute the same amount to ΔP_2 as to ΔD_2 , so $\Delta P_2 = \Delta D_2$, or, equivalently, $\Delta D_2 = \frac{1}{2}\Delta P_2 + \frac{1}{2}\Delta P_2$. Letting $c = 1$ if at least one *FinalCut* or *Merge-After-Cut* is executed and 0 otherwise, we thus have that $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}(\Delta P_1 + \Delta P_2) - 1 + \frac{1}{2}c + \frac{1}{2}\Delta P_2$.

By Claim 2, we know that $\Delta P_2 \geq 1$. Therefore, if $c = 1$, or if $\Delta P_2 \geq 2$, then $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}(\Delta P_1 + \Delta P_2)$. If neither of those holds, then the retroactive merge ensures that $|E(T'_2) \setminus E(\tilde{T}'_2)| = \Delta P_1 + \Delta P_2 - 1$, and so we again have $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}|E(T'_2) \setminus E(\tilde{T}'_2)|$. \square

6.2.2 Analysis of Procedure 4c

The following claim will be needed to show that if \hat{v}_1, \hat{v}_2 are active siblings in T'_2 , then the dual solution remains feasible when we set $y_{\hat{v}_1}$ to 1. The proof is deferred to Section A.

Claim 5. *If \hat{v}_1 and \hat{v}_2 are active siblings in T'_2 in line 84 of Procedure 4c, then one of them is \hat{b} .*

Lemma 9. *Let $(T'_1, T'_2, \mathcal{L}', y)$ be a valid tuple, that has been preprocessed by procedure PREPROCESS, and let $R \cup B$ be a minimal incompatible active sibling set with $\text{lca}_2(R) = \text{lca}_2(R \cup B)$, for which $\text{RESOLVESET}(R)$ returns FAIL. If, after executing lines 44–50, the three remaining active leaves are in two distinct trees in T'_2 , then the tuple $(\tilde{T}'_1, \tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y})$ after executing lines 44–50 followed by Procedure 4c is valid, and satisfies*

$$D(\tilde{T}'_2, \tilde{\mathcal{L}}', \tilde{y}) - D(T'_2, \mathcal{L}', y) \geq \frac{1}{2}(|E(T'_2) \setminus E(\tilde{T}'_2)|).$$

Proof. We first argue that the tuple after executing Procedure 4c has all the properties of a valid tuple, except possibly for dual feasibility. The only operation that is executed that we did not see before, is the handling of \hat{v}_1, \hat{v}_2 , which may not both be in R or B . However, we have cut off the only other active leaf in $R \cup B$, i.e., \hat{u} , in line 80 or line 82, and we thus know that \hat{v}_1, \hat{v}_2 are indeed active siblings in T'_1 when we merge them or execute $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$ in the next lines of the procedure.

By Lemma 6, we know that the dual solution is valid and $\{\hat{b}\}$ -safe and $\{\hat{r}_1, \hat{r}_2\}$ -safe at the start of the procedure. We now consider the dual solution, by looking at the different ways in which Procedure 4c may be executed. We let ΔP_2 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to Procedure 4c, and let ΔD_2 the change in the dual objective due to Procedure 4c.

- \hat{u} is cut off, followed by a merge of active sibling pair \hat{v}_1 and \hat{v}_2 .

As in lines 73–74 of Procedure 4b, the effect of cutting off \hat{u} on the dual solution is effectively zero if \hat{u} was the only active leaf in its tree in T'_2 , and otherwise we have deleted one edge from T'_2 and we increase the dual objective by 1. The load is increased by 1 only on sets L containing \hat{u} and another active leaf in the tree in T'_2 that contained \hat{u} . Note that $\hat{u} \in \{\hat{b}, \hat{r}_1, \hat{r}_2\}$, and the dual solution thus remains feasible, by the fact that the dual solution was $\{\hat{b}\}$ -safe and $\{\hat{r}_1, \hat{r}_2\}$ -safe.

We now consider the effect of merging \hat{v}_1 and \hat{v}_2 . Recall that $\hat{v}_1 = \hat{b}$ because of the relabeling and Claim 5. Assume without loss of generality that $\hat{v}_2 = \hat{r}_2$ and $\hat{u} = \hat{r}_1$. If A is the set of active leaves in the tree in T'_2 containing \hat{v}_1, \hat{v}_2 , then the dual is changed by decreasing z_A by 1, and increasing $y_{\hat{b}}$ and $z_{A \setminus \{\hat{b}\}}$ by 1. This increases the dual objective by 1, and it increases the load only on sets L containing both \hat{b} and a leaf in $A \setminus \{\hat{b}\}$. The load on such a set L must have been at most 0 at the start of the procedure, by the fact that the dual solution was $\{\hat{b}\}$ -safe.

Finally, we argue that the load on a compatible set cannot increase twice by the above: Suppose it did, then L must contain $\hat{u} = \hat{r}_1$ and \hat{b} and a leaf in $A \setminus \{\hat{b}\}$ and an active leaf in the tree in T'_2 that remains after cutting off \hat{u} . At the start of the procedure, it cannot have been the case that both the tree in T'_2 containing \hat{v}_1, \hat{v}_2 and the tree in T'_2 containing \hat{u} contained $\text{lca}_2(R \cup B)$. Hence, for one of these trees the leaves $x \notin B \cup R$ do not satisfy $\hat{r}_1 \hat{b} | x$ in T_2 , and thus, such a set L is not compatible. The dual solution therefore remains feasible.

We have that ΔP_2 is either 0 or 1, and $\Delta D_2 = 1 + \Delta P_2$.

- \hat{u} is cut off, followed by $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$, after which \hat{v}_1 is cut off.

Noting as before that executing line 80 does not effectively change the dual solution, we have three operations that potentially change the dual solution and increase the load on a set L :

- If line 82 is executed, let $A_{\hat{u}}$ be the active leaves of the tree in T'_2 containing \hat{u} before cutting off \hat{u} . Then $z_{A_{\hat{u}}}$ is decreased by 1, and $z_{A_{\hat{u}} \setminus \{\hat{u}\}}$ and $y_{\hat{u}}$ are increased by 1. The load is thus increased only on a set L if it contains \hat{u} , and some leaf in $A_{\hat{u}} \setminus \{\hat{u}\}$. The dual objective value increases by 1.
- To consider the effect of the load on a set L when executing $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$, note that \hat{v}_1, \hat{v}_2 are in the same tree in T'_2 and are not active siblings in T'_2 when executing $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$. Let \hat{v}_2 be the leaf that remains active after executing $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$, and let $A_{\hat{v}_1} \setminus \{\hat{v}_1\}$ be the set W that is cut off by line 13 of RESOLVEPAIR . From the discussion in the proof of Lemma 3, the load increases only for a set L containing \hat{v}_1 and a leaf in $A_{\hat{v}_1} \setminus \{\hat{v}_1\}$.
- When executing line 89, let $A_{\hat{v}_2}$ be the active leaves in the tree in T'_2 containing \hat{v}_2 before cutting off \hat{v}_2 . By Claim 2, $A_{\hat{v}_2} \setminus \{\hat{v}_2\} \neq \emptyset$. Then $z_{A_{\hat{v}_2}}$ is decreased by 1, and $z_{A_{\hat{v}_2} \setminus \{\hat{v}_2\}}$ and $y_{\hat{v}_2}$ are increased by 1. The load is thus increased on a set L if it contains \hat{v}_2 , and some leaf in $A_{\hat{v}_2} \setminus \{\hat{v}_2\}$. The dual objective value increases by 1.

By the fact that the dual solution was $\{\hat{b}\}$ -safe and $\{\hat{r}_1, \hat{r}_2\}$ -safe, the load on any set L that has its load increased by one or more of the above must have had load 0 at the start of the procedure. Suppose there is a set L that has its load increased by more than 1. Then L must contain two leaves from $\{\hat{r}_1, \hat{r}_2, \hat{b}\}$, and leaves in two of the sets $A_{\hat{u}} \setminus \{\hat{u}\}, A_{\hat{v}_1} \setminus \{\hat{v}_1\}, A_{\hat{v}_2} \setminus \{\hat{v}_2\}$ described above. Now, each of the sets $A_{\hat{x}} \setminus \{\hat{x}\}$ for $\hat{x} = \hat{u}, \hat{v}_1, \hat{v}_2$ are in the same tree in T'_2 at the end of the procedure, and at most one of these trees can contain $\text{lca}_2(R \cup B)$. Hence, at most one of these three sets can contain leaves that are not in an inconsistent triplet with two leaves in $R \cup B$. Hence, a set that has its load increased by more than 1 must be incompatible. The dual solution therefore remains feasible.

Let $c = 1$ if a *Merge-After-Cut* was performed in $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$ and $c = 0$ otherwise, and let $d = 1$ if line 82 was executed to cut off \hat{u} in T'_2 , and $d = 0$ otherwise. We then have that $\Delta P_2 = d + 2 + (1 - c)$: the “2” comes from the edge that was deleted from T'_2 to cut off $W = A_{\hat{v}_1} \setminus \{\hat{v}_1\}$ in $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$, and the edge that was deleted to cut off \hat{v}_2 .

We also have that $\Delta D_2 = d + 2$: if $d = 1$, then line 82 increases the dual objective by 1; $\text{RESOLVEPAIR}(\hat{v}_1, \hat{v}_2)$, when \hat{v}_1, \hat{v}_2 are in the same tree but not active siblings in T'_2 , increases the dual objective value by 1 (see Table 1); cutting of \hat{v}_2 from its tree in T'_2 increased the dual objective value by 1.

We thus have $\Delta D_2 = \frac{1}{2}\Delta P_2 + \frac{1}{2}(1 + c + d)$.

We now let ΔP_1 be the number of edges in $E(T'_2) \setminus E(\tilde{T}'_2)$ due to lines 44–50, and we let ΔD_1 be the total change in the dual objective value by lines 44–50. We let $c' = 1$ if at least one *FinalCut* or *Merge-After-Cut* was executed in lines 44–50. As in the proof of Lemma 8, we have that $\Delta D_1 \geq \frac{1}{2}\Delta P_1 - 1 + \frac{1}{2}c'$.

From the discussion above, in the case when the procedure merges \hat{v}_1 and \hat{v}_2 as an active sibling pair in T'_2 in line 86, then $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}\Delta P_1 - 1 + \frac{1}{2}c' + 1 + \Delta P_2 \geq \frac{1}{2}(\Delta P_1 + \Delta P_2)$.

Otherwise, we have

$$\Delta D_1 + \Delta D_2 \geq \frac{1}{2}\Delta P_1 - 1 + \frac{1}{2}c' + \frac{1}{2}\Delta P_2 + \frac{1}{2}(1 + c + d) \geq \frac{1}{2}(\Delta P_1 + \Delta P_2) - \frac{1}{2} + \frac{1}{2}(c' + c + d).$$

Therefore, if $c' + c + d \geq 1$, we have $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}(\Delta P_1 + \Delta P_2)$. Furthermore, we note that if $c' + c + d = 0$, then the condition in line 90 is satisfied, and thus the retroactive merge ensures in this case that $|E(T'_2) \setminus E(\tilde{T}'_2)| = \Delta P_1 + \Delta P_2 - 1$, and we again have $\Delta D_1 + \Delta D_2 \geq \frac{1}{2}|E(T'_2) \setminus E(\tilde{T}'_2)|$. \square

Combining Lemmas 5, 7, 8 and 9, and noting that the dual objective value is a lower bound on the objective value of the optimal solution, we have thus proved our main result:

Theorem 2. *Algorithm 2 is a 2-approximation algorithm for the Maximum Agreement Forest problem.*

7 Implementation Details

We implemented the Red-Blue approximation algorithm in Java, and tested it on instances with $|\mathcal{L}| = 2000$ leaves that were generated as follows: the number of leaves in the left subtree is set equal to a number between 1 and $|\mathcal{L}| - 1$ drawn uniformly at random, and a subset of this size is chosen uniformly at random from the label set. Then this procedure recurses until it arrives at a subtree with only 1 leaf — this will be the whole subtree.

After generating T_1 as described above, the tree T_2 was created by doing 50 random Subtree Prune-and-Regraft operations (where random means that the root of the subtree that is pruned was chosen uniformly at random, as well as the edge which is split into two edges, so that the new node created can be the parent of the pruned subtree, under the conditions that this is a valid SPR-operation). This construction allows us to deduce an upper bound of 50 on the optimal value. Our algorithm finds a dual solution that in 44% of the 1000 runs is equal to the optimal dual solution, and in 37% of the runs is 1 less than the optimal solution. The observed average approximation ratio is about 1.92. After running our algorithm, we run a simple greedy search algorithm which repeatedly looks for two trees in the agreement forest that can be merged (i.e., such that the resulting forest is still a feasible solution to MAF). The solution obtained after executing the greedy algorithm decreases the observed approximation ratio to less than 1.28. The code is available at <http://frans.us/MAF>.

8 Conclusion

We have shown how to construct an agreement forest for two rooted binary input trees T_1 and T_2 along with a feasible dual solution to a new LP relaxation for the problem. The objective value of the dual solution is at least half the number of components in the agreement forest. Since the objective value of any dual solution

gives a lower bound on the optimal value, this implies that our algorithm is a 2-approximation algorithm for MAF. This improves on the previous best approximation guarantee of 2.5 by Shi et al. [12].

Our algorithm and analysis raise a number of questions. First of all, although we believe that, conceptually, our algorithm is quite natural, the actual algorithm is complicated, and it would be interesting to find a simpler 2-approximation algorithm. Secondly, it is clear that our algorithm can be implemented in polynomial time, but the exact order of the running time is not clear. The bottleneck seems to be the finding of a minimal incompatible active sibling set, although it may be possible to implement the algorithm in a way that simultaneously processes sibling pairs as in RESOLVEPAIR, while it is looking for a minimal incompatible active sibling set.

Acknowledgements We thank Neil Olver and Leen Stougie for fruitful discussions.

References

- [1] Benjamin L. Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.
- [2] Maria Luisa Bonet, Katherine St John, Ruchi Mahindru, and Nina Amenta. Approximating subtree distances between phylogenies. *Journal of Computational Biology*, 13(8):1419–1434, 2006.
- [3] Magnus Bordewich, Catherine McCartin, and Charles Semple. A 3-approximation algorithm for the subtree distance between phylogenies. *Journal of Discrete Algorithms*, 6(3):458–471, 2008.
- [4] Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Ann. Comb.*, 8(4):409–423, 2004.
- [5] Charles Darwin. *Notebook B: Transmutation of species (1837–1838)*. In: John van Wyhe: The Complete Work of Charles Darwin Online, 2002. <http://darwin-online.org.uk/>.
- [6] Martin Farach and Mikkel Thorup. Optimal evolutionary tree comparison by sparse dynamic programming. In *FOCS '94: Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pages 770–779. IEEE, 1994.
- [7] Martin Farach and Mikkel Thorup. Sparse dynamic programming for evolutionary-tree comparison. *SIAM Journal on Computing*, 26(1):210–230, 1997.
- [8] A. D. Gordon. A measure of the agreement between rankings. *Biometrika*, 66(1):7–15, 1979.
- [9] Jotun Hein, Tao Jiang, Lusheng Wang, and Kaizhong Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71(1-3):153–169, 1996.
- [10] Estela M. Rodrigues. *Algoritmos para Comparação de Árvores Filogenéticas e o Problema dos Pontos de Recombinação*. PhD thesis, University of São Paulo, Brazil, 2003. Chapter 7, available at <http://www.ime.usp.br/~estela/studies/tese-traducao-cp7.ps.gz>.
- [11] Estela M. Rodrigues, Marie-France Sagot, and Yoshiko Wakabayashi. The maximum agreement forest problem: approximation algorithms and computational experiments. *Theoretical Computer Science*, 374(1-3):91–110, 2007.
- [12] Feng Shi, Qilong Feng, Jie You, and Jianxin Wang. Improved approximation algorithm for maximum agreement forest of two rooted binary phylogenetic trees. *Journal of Combinatorial Optimization*, 2015.
- [13] Mike Steel and Tandy Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, November 1993.

- [14] Chris Whidden, Robert G. Beiko, and Norbert Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.
- [15] Chris Whidden and Norbert Zeh. A unifying view on approximation and FPT of agreement forests. In *Algorithms in Bioinformatics*, volume 5724 of *Lecture Notes in Computer Science*, pages 390–402. Springer Berlin Heidelberg, 2009.
- [16] Yufeng Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25(2):190–196, 2009.
- [17] Yufeng Wu and Jiayin Wang. Fast computation of the exact hybridization number of two phylogenetic trees. In *Bioinformatics Research and Applications*, volume 6053 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2010.

A Deferred Proofs

Claim 2. *Just before line 72 and line 89, the leaf \hat{b} and \hat{v}_2 , respectively, is not the only active leaf in its tree in T'_2 .*

Proof of Claim 2: We consider T'_2 in line 51, and we note that \hat{r}_1, \hat{r}_2 and \hat{b} are contained in multiple trees in T'_2 . We show that, if \hat{r}_1, \hat{r}_2 are in the same tree in T'_2 , then that tree contains at least one active leaf w that is not a descendent of $\text{lca}_2(R \cup B)$. Otherwise, we show that the tree in T'_2 containing \hat{b} contains at least one active leaf w that is not a descendent of $\text{lca}_2(R \cup B)$. This suffices to prove the claim, since the leaf w will be in the same tree as \hat{b} if line 72 is executed, or in the same tree as \hat{v}_2 if line 89 is executed.

First, suppose \hat{r}_1 and \hat{r}_2 are in the same tree in T'_2 . Then this tree contains $\text{lca}_2(R \cup B) = \text{lca}_2(\hat{r}_1, \hat{r}_2)$. It then follows from Observation 4 (3) and the fact that lines 47–50 cannot delete an edge in T'_2 above $\text{lca}_2(R \cup B)$ that the tree containing \hat{r}_1, \hat{r}_2 contains a leaf w that is not a descendent of $\text{lca}_2(R \cup B)$.

Otherwise, suppose \hat{r}_1 and \hat{r}_2 are in different trees in T'_2 . We will show that this implies that \hat{b} must be in the same tree as $\text{lca}_2(R \cup B)$, and thus that the tree containing \hat{b} in T'_2 contains at least one leaf w that is not a descendent of $\text{lca}_2(R \cup B)$, by the same reasoning as above. First of all, note that if the tree containing $\text{lca}_2(R \cup B)$ contains at least one active leaf in B at the start of line 47, then the same holds after line 50: this is because otherwise there must be a *FinalCut* that cuts off the last leaf, say u , in B from this tree in a call to `RESOLVEPAIR`(u, v), with $v \in B$. But since $\text{lca}_2(u, v)$ is on the path from u to $\text{lca}_2(R \cup B)$, the tree containing u and $\text{lca}_2(R \cup B)$ contains $\text{lca}_2(u, v)$, and thus we would have relabeled u and v and cut off the other leaf in the pair, by line 2 of `RESOLVEPAIR`. Thus \hat{b} must be in the same tree as $\text{lca}_2(R \cup B)$, unless this tree contained no active leaves in B at the start of line 47. But in the latter case, no edges are cut from the tree containing $\text{lca}_2(R \cup B)$ by lines 47–50, and thus, using Observation 4 (2), \hat{r}_1 and \hat{r}_2 would still be in the same tree in T'_2 , contradicting our assumption. \square

Claim 3. *The leaves u', v' in line 76 and line 91 exist. Furthermore, the set W in line 76 and line 91 contains only active leaves that have the same color as \hat{r}_2 and \hat{u} , respectively.*

Proof of Claim 3: To simplify notation, we will let x denote \hat{r}_2 or \hat{u} respectively. Let X be the color of x , i.e., X is R or B . We say an active tree in T'_2 is X -bicolored tree if it contains active leaves in X and active leaves that are not in X , and we will say it is an X -unicolored tree if all its active leaves are in X . Consider the first moment when the tree containing x in T'_2 is X -unicolored, i.e., when all active leaves in the tree in T'_2 containing x are in X . There are three options: (1) this is already the case at the point where the current sets R and B were defined, i.e., in line 44, (2) this is achieved through a call to `RESOLVEPAIR` for an active sibling pair $\tilde{u}, \tilde{v} \in X$, and (3) this is achieved through a call to `RESOLVEPAIR` for an active sibling pair $\tilde{u}, \tilde{v} \notin X$.

We will show that if (1) or (2) happens, then Algorithm 2 must have executed at least one *FinalCut* or *Merge-After-Cut* in some call to `RESOLVEPAIR` in lines 44–50. Since this contradicts the condition in line 75, respectively line 90, it must be the case that (3) happens, and this is exactly what is stated in the claim.

For (2), note that a call to $\text{RESOLVEPAIR}(\tilde{u}, \tilde{v})$ where $\tilde{u}, \tilde{v} \in X$ are an active sibling pair in T'_1 can only change a X -bicolored tree into an X -unicolored tree if it cuts off a set of leaves that are not in X and what remains is a tree containing only active leaves in X , i.e., either line 4 in RESOLVEPAIR , or line 13 followed by line 15. Hence, in case (2) the call to $\text{RESOLVEPAIR}(\tilde{u}, \tilde{v})$ executes a *FinalCut* or a *Merge-After-Cut*.

For (1), note that the tree containing x in T'_2 in line 44 must contain at least one other active leaf in X , say x' , since x was not deactivated by the preprocessing. Since we also did not merge these leaves in PREPROCESS , they must not have been active siblings in T'_1 . Hence, there must have been some other tree in T'_2 in line 44 that contains $x'' \in X$ such that $x'x''|x$ in T_1 or $xx''|x'$ in T_1 . Repeating this argument if necessary shows that there exists an X -bicolored tree in T'_2 at the start of the while-loop. Furthermore, the argument also shows that there exists such an X -bicolored tree, say with active leaf set A , such that $\text{lca}_2(A)$ is a strict descendent of $\text{lca}_2(x, x')$, and hence also a strict descendent of $\text{lca}_2(R \cup B)$, in the original tree T_2 .

We now show that there must have been at least one *FinalCut* or *Merge-After-Cut* executed on the leaves in $A \cap (R \cup B)$.

First, suppose A contains leaves in R , i.e., the tree in T'_2 containing A at the start of the while-loop is R -bicolored. Note that $\hat{r}_1 \notin A, \hat{r}_2 \notin A$: when $\text{RESOLVESET}(R)$ fails, \hat{r}_1 and \hat{r}_2 are in the same tree in T'_2 , and hence, this must have also been true at the start of the while-loop. But since $\text{lca}_2(\hat{r}_1, \hat{r}_2) = \text{lca}_2(R \cup B)$, this means that \hat{r}_1, \hat{r}_2 cannot both be in A , as this contradicts the fact that $\text{lca}_2(A)$ is a strict descendent of $\text{lca}_2(R \cup B)$. But if A does not contain \hat{r}_1, \hat{r}_2 , then all leaves in $A \cap R$ will be inactive after executing $\text{RESOLVESET}(R)$. Hence, by Observation 3, at least one *FinalCut* or *Merge-After-Cut* must have been performed during $\text{RESOLVESET}(R)$.

Now, suppose A contains no leaves in R . Then X must be B . The execution of $\text{RESOLVESET}(R)$ does not affect this tree, and since the leaves x, x' are merged at some point during lines 47–50, the leaves in $A \cap B$ must all be deactivated by lines 47–50. Hence, we can again invoke Observation 3, at least one *FinalCut* or *Merge-After-Cut* must have been performed in lines 47–50. \square

Claim 4. *Let u' be the leaf and W the set identified in line 76 or line 91, respectively, and let L_1, \dots, L_k be the leaf sets of the trees in T'_1 at this moment, excluding the trees containing u' and W , and let $M_1, \dots, M_{k'}$ be the leaf sets of the trees in T'_2 at this moment, excluding the trees containing u' and W . Then, $V_1[L_j] \cap V_1[\{u'\} \cup W] = \emptyset$ for all $j = 1, \dots, k$ and $V_2[M_{j'}] \cap V_2[\{u'\} \cup W] = \emptyset$ for all $j' = 1, \dots, k'$.*

Proof of Claim 4: We first consider T'_2 . Let v' be such that the execution of $\text{RESOLVEPAIR}(u', v')$ cut off set W in T'_2 . Since no *Merge-After-Cut* was executed, we know that u' was also cut off by $\text{RESOLVEPAIR}(u', v')$. Note that before this execution of $\text{RESOLVEPAIR}(u', v')$, $p_2(u') = p_2(W)$; we will refer to this node as p . The edge below p was deleted to cut off W in line 13. Then, in line 17 the edge below the new parent of u' was deleted. Therefore, p is in the tree in T'_2 containing u' , and thus all nodes in T_2 in $V[\{u'\} \cup W]$ are either in the tree in T'_2 containing W or the tree containing u' . After $\text{RESOLVEPAIR}(u', v')$, the tree containing u' in T'_2 is inactive, and for the tree in T'_2 containing W , the active leaves are either all in R or all in B by Claim 3. Therefore, the executions of RESOLVEPAIR until we reach line 76 or line 91, respectively, do not affect the edge set of the tree in T'_2 containing W . Hence for the nodes in T'_2 we can conclude that $V_2[M_{j'}] \cap V_2[\{u'\} \cup W] = \emptyset$ for all $j' = 1, \dots, k'$.

Now consider T'_1 . Suppose, by means of contradiction that $V_1[L_j] \cap V_1[\{u'\} \cup W] \neq \emptyset$ for some L_j . We have the following claim.

Claim 6. *If $V_1[L_j] \cap V_1[\{u'\} \cup W] \neq \emptyset$, then there exist $x, y \in L_j$ that are active at the moment that $\text{RESOLVEPAIR}(u', v')$ was executed, such that $V_1[\{x, y\}] \cap V_1[\{u'\} \cup W] \neq \emptyset$ and there was some execution of $\text{RESOLVEPAIR}(x, y)$ after $\text{RESOLVEPAIR}(u', v')$ in which x and y are merged.*

First, we note that all leaves in L_j must be inactive, since all nodes in $V_1[\{u'\} \cup W]$ are descendants of $\text{lca}_1(R \cup B)$, and all leaves that are descendants of $\text{lca}_1(R \cup B)$ in T'_1 are in inactive trees. The moment that the last leaf in L_j was made inactive, we cut off the *subtree* in T'_1 represented by this leaf (i.e., the leaves that were (recursively) merged with it).

We now consider the active leaves excluding $W \cup \{u'\}$ just before the execution of $\text{RESOLVEPAIR}(u', v')$ and the subtrees in T'_1 that they represent. Note that these subtrees are not components of T'_1 , but that,

for a given active leaf u , the subtree represented by u is one of the two subtrees rooted at the children of $p_1(u)$, namely the subtree that contains u . By definition, these subtrees contain exactly one active leaf, and therefore these subtrees do not contain a leaf in $W \cup \{u'\}$. It follows that none of the nodes in $V_1[\{u'\} \cup W]$, are in such a subtree: Since all active leaves are in the same tree in T'_1 , and since the leaves in $\{u'\} \cup W$ are active, a subtree containing a node in $V_1[\{u'\} \cup W]$ would also contain either u' or W .

Hence, there must have been some moment, between the execution of $\text{RESOLVEPAIR}(u', v')$ and the moment of the retroactive merge, where some node in $V_1[\{u'\} \cup W]$ became part of the subtree represented by some active leaf, say y , and the only way in which the subtree represented by y can change is when some active leaf x is merged with y . Furthermore, the node in $V_1[\{u'\} \cup W]$ that becomes part of the subtree represented by y must be in $V_1[\{x, y\}]$. We have thus proven Claim 6.

We now show how Claim 6 can be used to prove Claim 4. First, suppose x, y are a different color from u', v' , i.e., if $u', v' \in R$, then $x, y \in B$ and vice versa. Note that then W has the same color as x, y , by Claim 3. Since $V_1[\{x, y\}]$ intersects $V_1[\{u'\} \cup W]$, and u' has a different color from W , we have that W is a descendent of $\text{lca}_1(x, y)$. But note that one leaf in W , either \hat{r}_2 or \hat{u} , is still active at the start of Procedure 4b and Procedure 4c. Hence, x and y cannot have been merged before the start of this procedure. Furthermore, if there was a merge in Procedure 4c (in line 86 or line 88) we would not have executed a retroactive merge, i.e., we would not have reached line 91.

We now consider the case that x, y have the same color as u', v' . Let U be the color of x, y, u', v' , i.e., $U = R$ if $x, y, u', v' \in R$ and $U = B$ otherwise. We let \tilde{U} be the leaves in U that are active just before executing $\text{RESOLVEPAIR}(u', v')$ that are descendants of $\text{lca}_1(x, y)$. We will use Observation 3 to show that there must have been a *FinalCut* or a *Merge-After-Cut* executed in a call to RESOLVEPAIR with arguments $\tilde{u}_1, \tilde{u}_2 \in \tilde{U}$, between time t_1 (just before the call to $\text{RESOLVEPAIR}(u', v')$) and t_2 (just after the call to $\text{RESOLVEPAIR}(x, y)$ in which x and y are merged).

Since \tilde{U} contains the active leaves in the subtree of T'_1 rooted at $\text{lca}_1(x, y)$ which is a subtree of the subtree of T'_1 rooted at $\text{lca}_1(U)$, executions of RESOLVEPAIR between time t_1 and t_2 are performed for active sibling pairs in T'_1 that are either both in \tilde{U} , or both in $U \setminus \tilde{U}$. Furthermore, the latter do not affect any part of T'_2 that is not below $\text{lca}_2(\tilde{U}) = \text{lca}_2(x, y)$, and we may therefore assume that all calls to RESOLVEPAIR between time t_1 and t_2 are performed for active sibling pairs $u, v \in \tilde{U}$. In fact, we may assume for simplicity that $\text{lca}_2(x, y)$ is the root of the tree in T'_2 containing x and y , since this does not affect the outcome of any call to RESOLVEPAIR between time t_1 and t_2 .

We let $A(x, y)$ be the leaves that are descendants of $\text{lca}_2(x, y)$ in the original tree T_2 , and that are active at time t_1 . Note that $u' \in A(x, y)$, since u' must be a descendent of $\text{lca}_1(x, y)$ by the condition that $V_1[\{x, y\}] \cap V_1[W \cup \{u'\}] \neq \emptyset$, and by the fact that $u', x, y \in U$ and U is compatible, u' must then also be a descendent of $\text{lca}_2(x, y)$. Also, note that u' and v' were active siblings in T'_1 when executing $\text{RESOLVEPAIR}(u', v')$, so also $v' \in A(x, y)$. Finally, this implies that also $W \subset A(x, y)$, since W is the set of active leaves that is cut off in line 13 of $\text{RESOLVEPAIR}(u', v')$. Note that W, u', v' are all in the same tree in T'_2 at time t_1 , and hence at this moment, there exists a tree in T'_2 containing active leaves $A \subseteq A(x, y)$ that is \tilde{U} -bicolored. At time t_2 , we have just merged x with y , and so the only leaf in \tilde{U} that is still active is y . Furthermore, by our simplifying assumption that $\text{lca}_2(x, y)$ is the root of the tree in T'_2 containing x and y , the tree in T'_2 containing y has no other active leaves, and is therefore \tilde{U} -unicolored. We have thus shown that at time t_1 , there is a tree in T'_2 with active leaf set A that is \tilde{U} -bicolored, and at time t_2 , all leaves in \tilde{U} (and hence in $A \cap \tilde{U}$) are inactive or in \tilde{U} -unicolored trees. Furthermore, we argued that we may assume that all calls to RESOLVEPAIR between time t_1 and t_2 are performed for active sibling pairs $u, v \in \tilde{U}$.

By Observation 3, there must therefore have been some call to RESOLVEPAIR that performed a *FinalCut* or *Merge-After-Cut* between time t_1 and t_2 . \square

Claim 5. *If \hat{v}_1 and \hat{v}_2 are active siblings in T'_2 in line 84 of Procedure 4c, then one of them is \hat{b} .*

Proof of Claim 5: Suppose by contradiction that \hat{v}_1 and \hat{v}_2 are \hat{r}_1 and \hat{r}_2 . First, suppose the tree containing \hat{r}_1, \hat{r}_2 contained some active leaf in B after $\text{RESOLVESET}(R)$. Since, at the start of Procedure 4c, \hat{b} is the only active leaf in B , and \hat{b} is not in the same tree as \hat{r}_1, \hat{r}_2 , it must be the case that some execution of $\text{RESOLVEPAIR}(u, v)$ with $u, v \in B$ cut off the leaf u , and after this operation the tree in T'_2 containing \hat{r}_1, \hat{r}_2

contains no active leaves in B . But note that, since $u, v \in B$, the path from u to $\text{lca}_2(R \cup B)$ must contain $\text{lca}_2(u, v)$, and thus, since $\text{lca}_2(\hat{r}_1, \hat{r}_2) = \text{lca}_2(R \cup B)$, the tree containing u, \hat{r}_1 and \hat{r}_2 before $\text{RESOLVEPAIR}(u, v)$ contained $\text{lca}_2(u, v)$. But this contradicts the condition in line 2 of RESOLVEPAIR for defining u if u and v are in different trees in T'_2 .

Hence, it must be the case that, if \hat{v}_1, \hat{v}_2 are \hat{r}_1, \hat{r}_2 , then the tree in T'_2 containing \hat{r}_1, \hat{r}_2 after $\text{RESOLVESET}(R)$ contained no leaves in B . Note that this immediately gives a contradiction, since then the tree containing \hat{r}_1, \hat{r}_2 was not changed after $\text{RESOLVESET}(R)$, and thus \hat{r}_1, \hat{r}_2 must have been active siblings at the moment when $\text{RESOLVESET}(R)$ failed, but this contradicts Observation 4 (4). Thus, \hat{v}_1 and \hat{v}_2 cannot be \hat{r}_1 and \hat{r}_2 , so one of them must be \hat{b} . \square